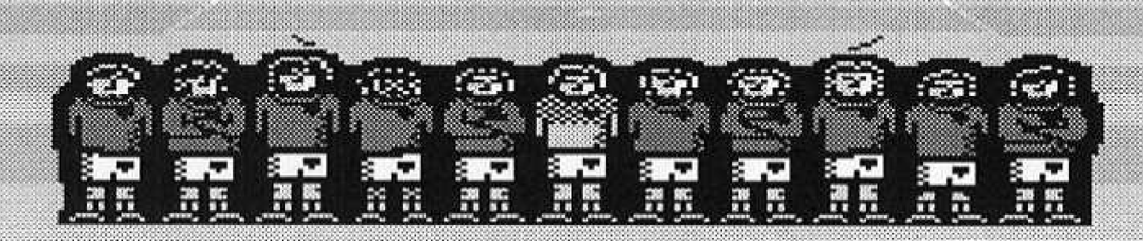




V.i.S.d.P.: Wolfgang Haller, Tel. 0221/680 33 10
Dabringhauser Strasse 141, 51069 Köln

E-mail: womoteam@t-online.de GEÄNDERT
Kölner Bank, **BLZ 371 600 87**, Kto-Nr. 7404 172 012

Ausgabe 149/150

Mai/Juni 2002

Das übliche Gesülze...

Wenn ihr dieses Heft in der Hand haltet, wird die Fußball-WM vorüber sein und wir alle wissen, ob wir Weltmeister, Vizeweltmeister oder Waldmeister geworden sind. Na, letzteres werden wir wohl nicht mehr, denn zur Zeit, wo ich dieses schreibe, stehen wir gesichert im Finale gegen Brasilien.

Ein wenig läßt sich aber vom Fußball auch auf unseren Club ableiten. Genau wie unser Team sind wir nicht mehr gerade weltbewegend, aber mithalten können wir allemale noch. Und wie in jeder Mannschaft bedarf es immer einiger Leistungsträger, bei uns sind das die Artikelschreiber fürs Info. Und einen Teamchef, der immer wieder darauf hinweist, das ihr 100% Leistung bringen müßt, um zu überleben. Diese leidige Aufgabe steht mir immer zu.

Nun, diese Ausgabe macht mir so richtig Freude. Endlich mal wieder etwas an Listings, das bringt doch den ein oder anderen mal wieder vor seinen 8-Bitter. Und wieder (schon zum drittenmal) 36 Seiten. Da braucht man keinen Artikel zu splitten oder zu kürzen.

Und noch etwas freut mich: 4 neue Mitglieder erhöhen unsere Mitgliederzahl auf 67. Da braucht das ZX-Team noch lange nicht zu überlegen, den SPC zu übernehmen (gell Willi vonne Küste!?!!).

Die Neuen im Club sind:

Jorge Canelhas (siehe auch Seite 31)
Apartado 3115, Miguel Bombarda
P-2745 Queluz, Portugal
☛ jcanelhas@retroreview.com
▭ http://www.retroreview.com

Martijn Groen
Tollensstraat 38, NL-3131 GZ Vlaardingen
☛ mgreen@wanadoo.nl
▭ http://mnemotech.ucam.org/download.html (SAM: B-DOS 1.7d)

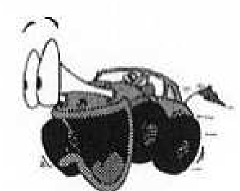
Mirko Seidel
Neenstetter Straße 20, 89183 Breitingen
☛ micro20000@hotmail.com

Robert van der Veeke
Ruysdealstraat 21D, NL-3141 HP Maassluis
☛ RJV.Veeke.NL.Sam.Coupe@caiw.nl
▭ http://home.kabelfoon.nl/~rjvveeke/Coupe/index.html

Zu vermelden gibt es noch, das sich Johan Koelmans email-Adresse geändert hat. Bitte schreibt ihm zukünftig unter:
☛ koelman28@zonnet.nl

Doch nun genug gesülzt. Erfreut euch am vorliegenden Heft und macht euch Gedanken, ob ihr nicht auch was dazu beitragen könnt. In diesem Sinne. (Wo)

Termine 2002



13.-14. Juli 2002
Das Z-Fest 2002! Jeweils von 10.00 Uhr bis open end im Dorfgemeinschaftshaus Fuldatal-Knickhagen (nördl. Kassel)

14./15. September 2002
Treffen in Wittenberg. Anmeldungen ab sofort bei: Norbert Opitz
Joh.-Friedrich-Böttger-Straße 7
06886 Wittenberg
Email: NorbertOpitz.Wittenberg@t-online.de

21. September 2002, 10-16 Uhr
Treffen in Bunnik.

5. Oktober 2002
Klubtreffen der JOYCE-User-AG e.V.

16. Oktober -18. Oktober 2002
Multiplatform Party called "Syndeecate".
Party place : Czech Republic, Roznov pod Radhostem, 1.hostinsky pivovar
Entrance fee : not known. Main Computer Platforms : PC, Amiga, ZX Spectrum. Sleeping: your own sleeping-bag on the party place. How to eat : restaurant/pub is integrated! :-)
Questions? Write to "tdm@quick.cz"

Teilt mir bitte weiterhin alle euch bekannten und interessanten Termine mit.



ZX9200: neue Version

Den Spectrum-Emulator für das Nokia 9210 "ZX9200" (von: ***http://www.wildpalm.co.uk***) gibt es seit kurzem in neuer Version v1.1, die unter anderem folgende Neuerungen brachte:

- man kann nun auch (im .Z80-Format) abspeichern.
- die Geschwindigkeit der Emulation wurde erhöht.
- es wurde eine PAUSE-Funktion hinzugefügt, und wenn man die aufruft, erscheint eine Hilfs-Grafik, die die Tastanbelegung der Spectrum-Tastatur zeigt.
- der emulierte Joystick wird automatisch auf den im .Z80-File gespeicherten Modus umgestellt.
- Unterstützung des französischen 9210-Keyboard-Layouts.

Noch mehr Spectrum-Emulation News

Unter
http://classicgaming.dcemulation.com/
findet man jetzt die "finale" Version 1.0 des Spectrum Emulators "DreamSpec" für die Sega Dreamcast. Außerdem kann man dort nun auch den "DreamSpec" in einer Version für die Playstation 2 (=PS2) laden.
Diese Version beinhaltet jetzt die Unterstützung des BRIGHT-Attributes und auch erstmals Sound-Unterstützung.

Gruß! Luzie

ZXVGS.29 für Pentagon

Hello, this email message is a notification to let you know that a file has been uploaded to the Files area of the zxvgs group.

File: /ZX029PEN.zip
Uploaded by: yarek@sp7.zsk.p.lodz.pl
Description: ZXVGS 0.29 for Pentagon (with QTRANS)

You can access this file at the URL:

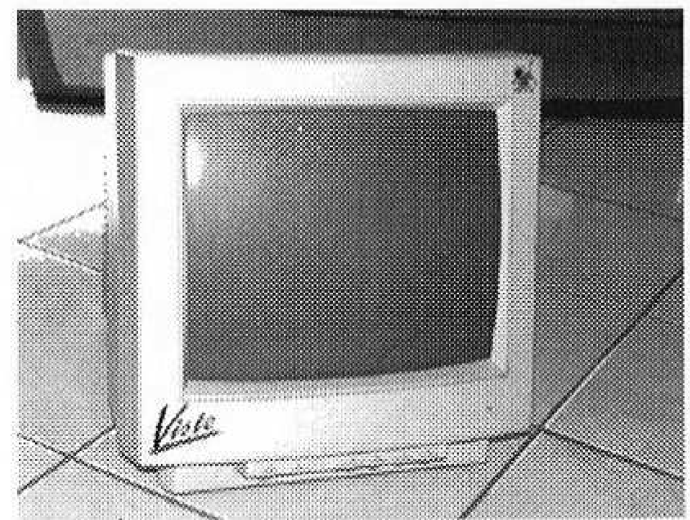
http://groups.yahoo.com/group/zxvgs/files/ZX029PEN.zip

Regards,
yarek_com <yarek@sp7.zsk.p.lodz.pl>



Hallo Wolfgang,

dass Tagebuch eines Chaoten ist echt toll und sollte weiter Bestandteil des Infos bleiben. In der Anlage sende ich eine kleine Liste von Artikel die ich mehrfach habe oder nicht weiter benötige:



1 Visto RGB Monitor (siehe Foto) für 26,- €
1 ZX Interface 1 (defekt) 2,- €
1 Drucker Star LC 10 mit extra neuem Druckerband gegen Gebot!

Viele nette Grüsse *Günther Marten*
guenther.marten@nwn.de

ZX<->PC Interface

Zu diesem Interface brauchte ich einen E-PROM-Eraser. Im Internet hatte ich eine Anzeige für eines gesetzt. Inzwischen konnte ich eines ersteigern, es kann also weitergehen.

Entwicklung eines neuen ZX81 Emulators

Im Z80-Emulator kann man mit einem geänderte ROM anfangen. Ich habe das ROM für RST 0 geändert, damit ich die Idee des neuen Emulators programmieren kann. Der neue Emulator wird wie der Videopac-Emulator funktionieren.
Der Original-Code wird zu einem Z80-Code emuliert und „übersetzt", beim 2. Lauf wird das übersetzte Programm „RUN"nen. Weil der ZX81 schon einen Z80 hat werden nur die Problem-Codes in RST 0 geändert. RST 0 wird dann die richtige Emulation machen.

Ein Beispiel

```
#8500     LD  A,10
#8502     LD  HL,#5000
#8505     LD  (HL),A
#8506     INC HL usw....
```

wird zu:

```
#8500     LD  A,10
#8502     LD  HL,#5000
#8505     RST 0
#8506     INC HL usw....
```

In einer Tabelle wird der Originalwert von #8505 gespeichert und durch RST 0 gut emuliert. Da die meisten Codes schneller als am ZX81 laufen (2 Mhz gegen 3,5 Mhz) wird am Ende eine gute Geschwindigkeit herauskommen.
Mit dem Z80-Emulator kann ich also ohne Hardware schon den Emulator bauen (aber nur mit 8K RAM). Am ZX Spectrum wird dafür ein Hardware-Interface mit 16K RAM auf die ROM Adressen gebaut damit sofort auch ein 16K ZX81 emuliert wird.

Dr Beep.

Hurra, es ist geschafft!

Speichererweiterung des MB 02 auf 512 kB

Auf dem Treffen in Urmond (NL) haben mir Roelof Koning und Johan Koelman von den holländischen Specci-Freunden geholfen, die Speichererweiterung für das MB 02 zu realisieren. Meine Mithilfe hat sich auf die Idee, die IC-Beschaffung und mein MB 02 als Versuchsobjekt beschränkt.
Für den Nachbau werden drei IC's benötigt. Den Speicher-IC und je ein 74LS00 und 74LS14. Die IC's habe ich 15mal vorrätig. Der Speicher-IC kommt in die in die Fassung auf der Leiterplatte (drei Pins hoch gebogen), 74LS14 direkt neben dem Speicher und der 74LS00 über einem beliebigen Nachbar-IC.
Es werden mitgeliefert:

- Schalt- und Lageplan
- Farbausdruck des geöffneten MB 02-Bildes
- Systemdiskette (Orginal 128kB-Diskette hat nicht funktioniert)

Der Preis wird bei etwa 25 Euro liegen.

Der Einbau müsste von jedem zu machen sein, der mit einem Lötkolben umgehen kann, denn nur der 74LS00 in „fliegender Verdrahtung" und wenige Punkte auf der Leiterplatte und drei Pins des Speicher-IC's müssen gelötet werden.
Wer Interesse an der Speichererweiterung hat und sich den Einbau zutraut, wendet sich an mich und ich werde es umgehend zuschicken. Wer sich den Einbau nicht zutraut, dem hilft vielleicht ein Elektronikfreund oder die nächsten Specci-Treffen werden noch mehr zum Basteltagen.

Meine Adresse ist:

Norbert Opitz
J. F. Böttger Str. 7
06886 Lutherstadt Wittenberg
Tel.: 03491 401573 (werktags nach 16 Uhr)
Mail: NorbertOpitz.Wittenberg@t-online.de

...mierter Schuss ins Blaue

menschlicher Einfluss ist nach dem Anpfiff nicht erlaubt.

„30000 Euro kostet jeder unserer Spieler", sagt Professor Thomas Christaller, Schölls Chef. Auch darum ist der *RoboCup* keineswegs ein reiner Zeitvertreib für durchgeknallte Schrauber, Tüftler und Hacker. Auf dem Fußballfeld kann die Wissenschaft beweisen, wie weit sie mit der Entwicklung mobiler, selbstständiger und intelligenter Maschinen ist.

Die Daten, die in den Spielern gesammelt werden, bringen die Forscher wieder ein Stück näher ans Ziel. Das kann der Bau von Robotern sein, die Unfallopfer bergen, fremde Planeten erforschen oder einfach nur bessere Autos bauen. „Mit Fußball nehmen wir den Leuten die Angst vor der Robotik", meint Schöll. „Roboter, die bügeln oder Einkaufen gehen, hätten sicher größere Akzeptanzprobleme."

Der Bonner Informatiker ist im AIS-Team für die Simulation des Techno-Kicks verantwortlich. Was die Hitzfelds und Toppmöllers mit Kreide auf die Tafel kritzeln, programmiert Schöll im Computer: Lauf- und Passwege, Abwehr- und Angriffsverhalten, aber auch Energiebedarf und mögliche Empfangsstörungen werden auf dem PC-Monitor durchgespielt.

„Die Abseits-Regel habe ich erst beim Roboter-Fußball kennengelernt", bekennt der gebürtige Schwabe, der sich seinen ersten Computer mit zwölf kaufte. „Ein Sinclair ZX 81 mit einem KB Arbeitsspeicher für 1700 Mark, sponsored by Oma."

Während andere Dreikäsehochs draußen dem runden Leder nachgelaufen sind, programmierte der kleine Peter die ersten simplen Tennisspiele im Kinderzimmer. Dass sein kickendes Bier-kisten-Quartett auf dem 10 x 5 Meter großen Feld oft so desorientiert wirkt, als habe er es mit dem Sinclair programmiert, weist Schöll entrüstet zurück: „Wenn's chaotisch wird, liegt das meist an den Handys, die hier im HNF benutzt werden. Die verwirren unsere Mannschaft."

Spaßige Show vor wissenschaftlichem Hintergrund: 6000 Fans verfolgten die knuffigen Ballartisten beim dreitägigen *RoboCup* in Paderborn. Tüftler Peter Schöll ist trotz Halbfinal-K.O. seines Teams zufrieden: „Die Holländer schießen einfach zu hart"

Immerhin: Eigentore gibt es selten, Kantersiege (8:1, 7:0) schon häufiger. Blau oder gelb müssen die Torflächen aussehen, sonst sehen die auf die beiden Schwedenfarben geeichten Torjäger schwarz.

Nicht nur im richtigen Fußballer-Leben, auch bei den Robotern dominieren in diesem Jahr die Werksteams. Sie sind in die Domäne der Universitäten und Forschungsinstitute eingebrochen. Die schussgewaltigen Philips-Schränke sind für Schölls Mannschaft im Halbfinale zu stark. Es reicht in der Middle Size League, der Königsklasse, zu Platz drei. Ein toller Erfolg für AIS und Peter Schöll, der für seine Diplom-Arbeit einen sechsrädrigen Roboter entwickelt hatte, „der nix konnte, außer nicht vor die Wand zu fahren."

Da können seine fantastischen Vier heute deutlich mehr. 31 Mal trafen die quirligen AIS-Bierkästen in acht Partien ins Blaue. Das lässt sachte Weltmeisterschafts-Träume reifen: „Mit unserem Klasse-Torwart ist im Juni in Fukuoka das Viertelfinale drin."

Das wird jedoch bloß ein Intermezzo sein. 2050 sollen humanoide Roboter den amtierenden Fußballweltmeister schlagen. Peter Schöll ist dann 82 Jahre alt. Wir nehmen den Fall auf Wiedervorlage.

Heiko Schlierenkamp

Info: www.robocup2002.org
www.ais.fraunhofer.de/dfg-robocup

Pfui!

13 prisma

Geschriebener Stuss für den Ofen!

Wer die Prisma als Beilage zu einer Tageszeitung erhält, dürfte sich beim Lesen der Ausgabe 18/2002 als Sinclair-Freund gewundert haben: Nicht über den damaligen Preis eines ZX81, sondern über den Kommentar eines offensichtlich desorientierten Schreibers! Dennoch, irgendwie kann man es ja auch als Kompliment auffassen, wenn man bedenkt, wo die Wurzeln liegen...

Sprite-Programmierung auf dem ZX Spectrum

1. Einleitung

In diesem Artikel geht es darum, wie man Sprites auf dem ZX Spectrum in Assembler-Sprache programmiert - Assembler-Grundkenntnisse werden dabei vorausgesetzt. Einige von euch werden sich jetzt sicher erst mal fragen „Was sind Sprites denn überhaupt?" - jeder der schon einmal vor einem Computerspiel gesessen ist, hat sie bereits gesehen! Es sind all die beweglichen Objekte wie z.B. die Spielfigur, ein Geschoß oder eine Aufzugsplattform. Der Begriff „Sprite" kommt aus dem Englischen und heißt soviel wie „Elfe" oder „Kobold". Sprites existieren schon seit es die Computerspiele gibt, der Name „Sprite" an sich ist aber vermutlich erst durch den Commodore 64 populär geworden und bezeichnet dort ein von der Hardware bereitgestelltes bewegliches Objekt, das von den Daten im Bildspeicher unabhängig ist (!) und vom Videochip nachträglich vor oder hinter das Bildsignal gelegt wird. Für Computer die leider nicht über diese doch recht praktische Hardware verfügen (wie z.B. den Schneider CPC) gibt es die „Software-Sprites", sie werden manchmal auch „Shapes" oder „BOBs" genannt („Blitter OBjects" - der Blitter ist ein Spezialchip des Amiga zum schnellen Verschieben von Speicherbereichen). Sie müssen mit Prozessor-Power in den Bildspeicher geschrieben werden, was sie natürlich langsamer macht.

Da „Uncle Clive" unseren Speccy ebenfalls nicht mit einer Sprite-Hardware bedacht hat (dadurch wäre der Speccy sicher auch deutlich teurer geworden), müssen wir uns auch hier per Software (?Shapes) behelfen, was angesichts des komplexen Bildspeicher-Aufbaus nicht ganz so einfach ist. Die Sprites sollen sich dann ja auch noch möglichst flimmerfrei und schnell genug bewegen - deswegen kommt man um die Maschinensprache kaum herum, was die Sache weiter verkompliziert. Weil viele bisher aus genau den Gründen kapitulierten, habe ich diesen Artikel geschrieben.

2. Der Bildspeicher („Screen") des ZX Spectrum

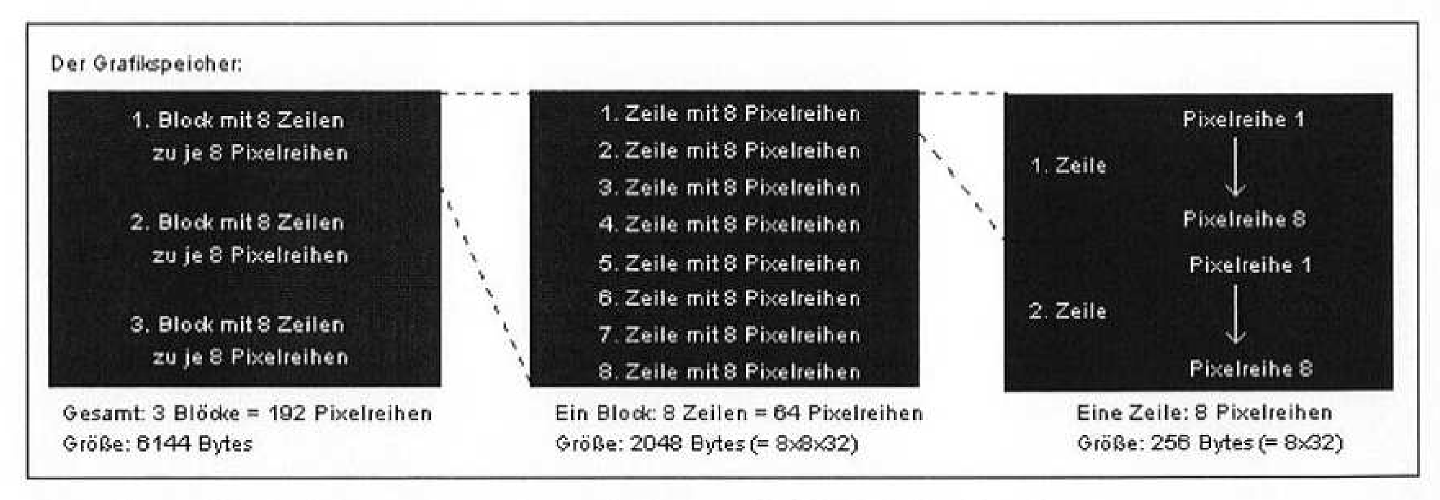
Bevor wir zu der Sprite-Routine kommen, erkläre ich erst einmal den Bildspeicher-Aufbau. Beim ZX Spectrum beginnt der Bildspeicher ab Adresse 16384 (4000 hex), er ist insgesamt 6912 Bytes groß und besteht (wie beim Commodore 64) aus einem getrennten Grafik- und Farbattribut-Bereich. Diese Trennung war damals, als Speicher-Chips noch sehr teuer waren, ein gern eingesetzter Trick, um den Speicherbedarf für ein Bild klein zu halten. Der Nachteil daran ist, daß man die Farbe für jeden einzelnen Pixel (= Bildpunkt) nicht frei wählen kann, sondern eine aus der Palette auswählen muß, die das Attribut, in dem er liegt, zur Verfügung stellt (beim Speccy 2 Farben je Attribut, beim C64 max. 4). Zum Vergleich: Ein „echter" Bildspeicher, d.h. mit 16 frei wählbaren Pixel-Farben und in Spectrum-Auflösung, wäre 24576 Bytes groß gewesen (s.h. Sam Coupé Mode 4) – einen 16K Spectrum hätte es so nie gegeben und beim 48K Spectrum wäre bereits der halbe Speicher belegt! Ihr seht also, die Trennung macht durchaus Sinn – und sie verhilft dem Spectrum zu dem in seiner Klasse (C64, Schneider CPC) kleinsten (und somit am schnellsten vom Prozessor füllbaren) Grafikspeicher !

Hier noch ein paar Daten zu dem Grafik- und Farbattribut-Bereich:

1. Der Grafik-Bereich. Er beginnt bei Adresse 16384 (4000 hex), endet bei 22527 (57FF hex) und ist 256 x 192 Pixel groß. Vom Basic aus können aber nur 256 x 176 Pixel erreicht werden. Jedes Pixel kann hier nur „an" (Wert 1) oder „aus" (Wert 0) sein. Daraus ergeben sich 8 Pixel pro Byte, also 32 Byte pro Zeile; mal 192 Zeilen ergibt das einen Speicherbedarf von 6144 Bytes.

2. Der Farbattribut-Bereich. Er beginnt bei Adresse 22528 (5800 hex) und endet bei 23295 (5AFF hex). Jedes Attribut bestimmt die Vorder- und Hintergrund-Farbe (und Blinken + Helligkeit) für ein 8 x 8 Pixel großes Quadrat auf dem Bildschirm.
   Der Bildschirm ist 32 x 24 Quadrate (= Chars bzw. Zeichen) groß, das ergibt einen Speicherbedarf von 768 Bytes. Aufbau eines Attribut-Bytes: [fbpppiii] mit f=Flash, b=Bright, p=Paper (3 Bits), i=Ink (3 Bits).

Wie oben bereits angedeutet, ist der Aufbau des Bildspeichers recht komplex. Warum das so ist, kann ich auch nicht sagen, aber vielleicht konnte so die Hardware einfacher konstruiert werden (andere Vorschläge ?). Während der Farbattribut-Bereich schön linear aufgebaut ist (Zeichen für Zeichen, Zeile für Zeile), setzt sich der Grafik-Bereich aus drei in sich abgeschlossenen rechteckigen Blöken mit einer Größe von je 32 x 8 Zeichen (Chars) zusammen (siehe Abbildung links und Mitte). Abgeschlossen deswegen, weil der nächste Block erst dann erreicht wird, wenn der aktuelle komplett durchlaufen wurde. Innerhalb eines Blocks geht es noch komplizierter zu – er ist in 8 ein-Zeichen-tiefe Zeilen unterteilt, weil die Pixel-Reihen einer Zeichen-Zeile nicht direkt aufeinander folgen, sondern einen Sprung von 8 Pixel-Reihen machen - d.h. nach der 1. Pixel-Reihe der ersten Zeile folgt die 1. Pixel-Reihe der zweiten Zeile und nicht (wie erwartet) die 2. Pixel-Reihe der ersten Zeile ! Erst nachdem alle acht 1. Pixel-Reihen durchlaufen wurden, kommt die 2. Pixel-Reihe der 1. Zeile und alle weiteren 2. Pixel-Reihen (siehe Abbildung Mitte und rechts)!



Das klang jetzt sicher etwas kompliziert, deswegen schaut ihr euch das Ganze am Besten mal mit folgendem Programm direkt auf dem Spectrum an. Es füllt den Grafikspeicher einfach von seinem Anfang bis zum Ende. Achtet genau auf den Pixel-Reihen-Sprung und darauf, wann ein ganzer Block gefüllt ist:

```
10 INK 0: PAPER 7: BORDER 7: FLASH
   0: BRIGHT 0: CLS: BORDER 2
20 FOR n=16384 TO 22527
30 POKE n,255
40 NEXT n
50 PAUSE 100
```

3. Die Sprite-Routinen

Nachdem ihr euch jetzt (hoffentlich) vorstellen könnt, wie der Bildspeicher aufgebaut ist, kommen wir zu der Sprite-Programmierung (farbige Sprites, d.h. mit Attributen, behandle ich in diesem Artikel allerdings nicht).

Dazu erst einmal etwas Theorie:

Ein normaler Sprite ist, wie gesagt, ein kleines rechteckiges Objekt, für dessen Breite man aus praktischen Gründen Bytes statt Pixel nimmt. Ganze Bytes kann man leichter und vor allem schneller als einzelne Pixel im Bildspeicher setzen - und die Geschwindig-

keit ist sehr wichtig! Beim Spectrum passen 8 Pixel in ein Byte, deswegen nehmen wir für die Sprite-Breite hier immer ein Vielfaches von 8 Pixeln. Dementsprechend wird der Sprite (zumindest bei den hier vorgestellten einfachen Sprite-Routinen) nicht mit Pixel- sondern nur mit Byte-Koordinaten auf dem Bildschirm plaziert (d.h. das Koordinaten-System besteht nicht aus 256 x 192 Einheiten, sondern nur aus 32 x 192). Um den Sprite auch horizontal pixelgenau zu plazieren, müßte man die Sprite-Daten rotieren, bevor man sie in den Bildspeicher kopiert - weil das aber für das Verständnis von Sprite-Routinen erst mal nicht so wichtig ist, verzichte ich hier darauf.

Für die folgenden Sprite-Routinen nehmen wir an, unser Sprite wäre 32 Pixel breit (32/8 = 4 Bytes) und 20 Pixel (= 20 Bytes) hoch. Die Sprite-Grafik liegt irgendwo außerhalb des Screens im Speicher und belegt dort 80 aufeinander folgende Bytes. Diese müssen nun richtig in den Bildspeicher kopiert werden, um den Sprite anzuzeigen. Zur Erinnerung, jede Zeile im Bildspeicher ist 32 Bytes breit und das komplette Bild ist 192 Bytes hoch – da der Sprite aber nur 4 Bytes breit ist, können wir also nicht einfach hergehen und jede Bildzeile komplett bis zum Ende mit den Sprite-Daten aus dem Speicher füllen, sondern müssen nach dem Schreiben von den 4 Sprite-Bytes in einer Pixelreihe für die nächsten 4 Sprite-Bytes in die nächst-tiefere Pixelreihe springen (diese Sprungdistanz nennt sich „Offset", hier 32–4 = 28). Nach dem Schreiben von 20 solcher Pixelreihen ist der Sprite komplett auf den Bildschirm kopiert und wir sind fertig.

Es ist sehr wichtig, daß wir nach dem Schreiben der 4 Bytes in einer Pixelreihe jeweils in der direkt darunterliegenden Pixelreihe weitermachen und nicht (wie beim Pixelreihen-Aufbau des Grafikspeichers) erst alle 1. Reihen, dann alle 2. Reihen, dann alle 3. Reihen, usw. füllen – das wäre zwar deutlich einfacher zu programmieren, der Sprite würde aber in diesem Fall stark flimmern! Die Schwierigkeit der Sprite-Programmierung auf dem Spectrum liegt also darin, trotz des komplizierten Grafikspeicher-Aufbaus immer die richtige Anfangsadresse der nächst-tieferen Pixelreihe für die Sprite-Daten zu berechnen - und das möglichst schnell.

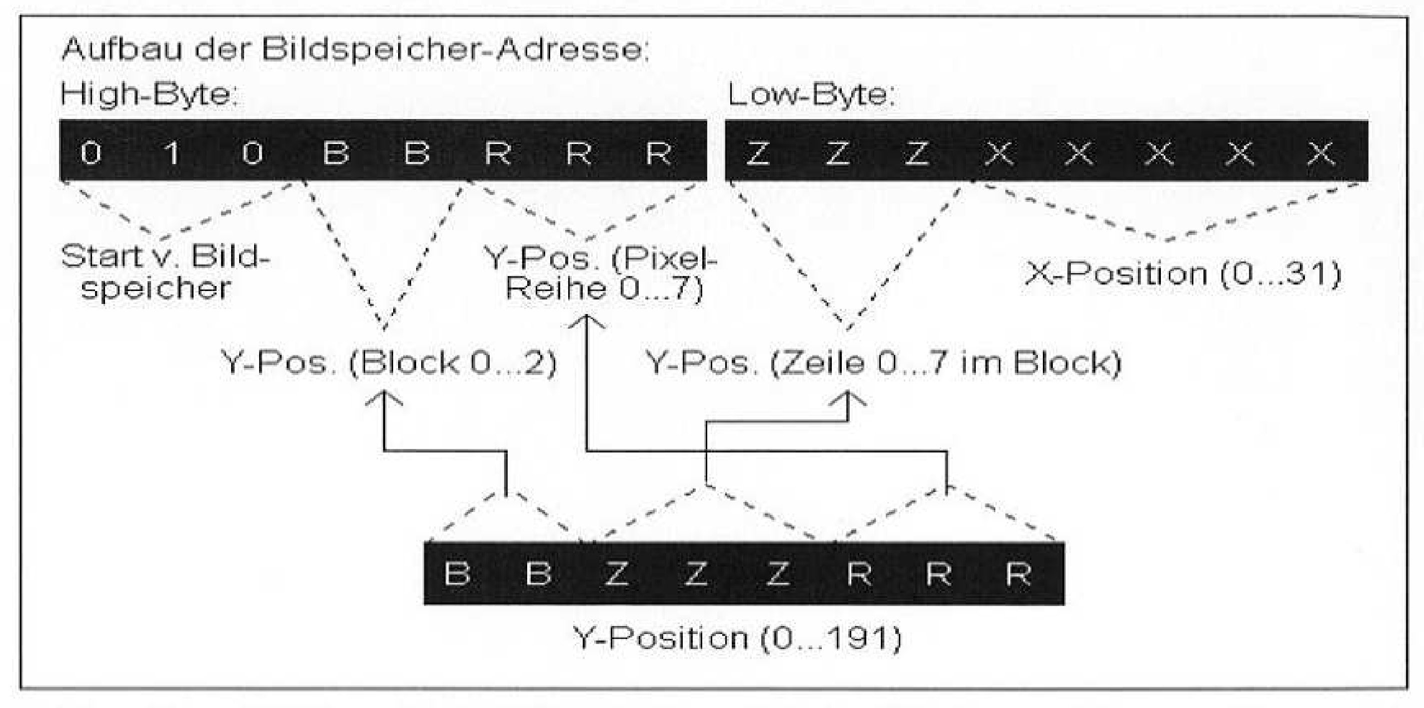
3.1. Die Berechnung der kompletten Bildadresse

Für die Darstellung von Sprites brauchen wir zwei Routinen. Die erste berechnet aus den von uns angegebenen Koordinaten (32x192) die komplette Bildadresse, ab der wir beginnen, den Sprite in den Bildspeicher zu kopieren. Die zweite ist einfacher und berechnet während dem Schreibvorgang immer nur die Adresse für das nächste Sprite-Byte. Diese Trennung ist notwendig, weil die erste Routine zu viel Zeit kostet, um sie häufig aufzurufen. Die zweite Routine wird zusammen mit der ersten Sprite-Routine bei 3.2 besprochen.

Noch ein wichtiger Hinweis: Wenn man für den Sprite eine X-Koordinate wählt, die zu weit am rechten Rand ist, dann gibt es (ähnlich wie bei einer Textverarbeitung) einen Zeilen-Umbruch, d.h. der Anfang des Sprites wird am rechten Rand dargestellt, der Rest taucht aber am linken auf! In unserem Beispiel sollte man keinen X-Wert größer als 28 verwenden. Ähnlich sieht es mit der Y-Koordinate aus, nur daß hier bei einem zu großen Wert (im Beispiel größer als 172) der Farbattribut-Bereich oder schlimmer noch der dem Bildspeicher folgende Haupt-Speicher überschrieben wird – das kann zu einem Absturz des Computers führen (vor allem beim 128er, bei dem nach dem Bild nicht der Druckerpuffer, sondern wichtige Daten folgen), also Vorsicht ! Fehlerhafte Spriteroutinen führen meist zu dem selben Ergebnis, deshalb sollte man vor einem Testlauf immer sein Programm speichern.

Mit dieser Routine kann man die komplette Bildadresse aus einem X- und Y-Wert er-

mitteln. Sie ist aufwändiger (langsamer) und wird nur einmal zur Berechnung des Sprite-Anfangspunkts benötigt. Um die X-Koordinate (Wert zwischen 0 und 31) umzusetzen, braucht man nicht groß zu rechnen, man kann sie einfach zur Bildspeicher-Anfangsadresse (links oben = 16384 / 4000 hex) hinzuaddieren. Den Additionswert für die Y-Koordinate (Wert zwischen 0 und 191) muß man aber entweder aus einer Tabelle auslesen oder selbst berechnen. Die Tabellen-Methode ist natürlich deutlich schneller (Tabellen werden allgemein gerne zur Geschwindigkeits-Optimierung bei Demos eingesetzt), hier beschreibe ich aber erst mal die Berechnungs-Methode.



Auf der obigen Abbildung ist die Y-Koordinate als Byte am unteren Rand dargestellt und die Bildadresse, die wir berechnen wollen, als 16 Bit-Wert (High- und Low-Byte) am oberen Rand. Die untersten 5 Bits des Low-Bytes der Bildadresse ergeben sich einfach aus der X-Koordinate, die Y-Koordinate dagegen muß wegen des komplizierten Bildspeicher-Aufbaus in die von oben bekannten drei Teile (Pixelreihe, Zeile, Block) zerlegt werden, bevor sie in die Bildadresse eingetragen werden kann.

Die kleinste Einheit des Bildspeichers ist die Pixelreihe - und weil es davon 8 pro Zeichen-Zeile (= nächstgrößere Einheit) gibt, nehmen wir die ersten 3 Bits (von rechts) der Y-Koordinate. Um von einem Byte im Bildspeicher zu dem genau darunterliegenden zu kommen, muß man den Wert 256 zu seiner Bildadresse hinzuaddieren - daraus folgt, daß wir unsere 3 Bits als unterstes in das High-Byte der Bildadresse eintragen müssen, weil 256 = 2^8 = achtes Bit von rechts bei Zählung ab 0 (s.h. „R"s in der Abbildung).

Die mittlere Einheit des Bildspeichers ist die Zeichen-Zeile. Von ihr gibt es auch 8, bevor die nächstgrößere Einheit (= der Block) folgt. Wir nehmen also weitere 3 Bits von der Y-Koordinate (die „Z"s auf der Abbildung). Um von einer Zeichen-Zeile zur nächst-tieferen zu gelangen, muß man den Wert 32 zu ihrer Bildadresse hinzuaddieren. Wir müssen unsere 3 Bits also direkt oberhalb der X-Koordinate in das Low-Byte der Bildadresse eintragen, weil 32 = 2^5 = fünftes Bit von rechts (immer bei Zählung ab 0).

Zum Schluß bleibt noch der Block-Anteil der Y-Koordinate übrig (die „B"s auf der Abbildung). Das sind die verbliebenen 2 Bits (für 3 Blocks). Um von einem Block zum nächsten zu gelangen, muß man den Wert 2048 (= 8 Zeilen * 8 Pixelreihen * 32 Bytes pro Zeile)

zu der Bildadresse hinzuaddieren. Unsere restlichen 2 Bits der Y-Koordinate müssen also bei 2048 = 2^11 = an der 11ten Stelle von rechts in die Bildadresse eingetragen werden. Das ist im High-Byte direkt oberhalb der Stelle, wo wir die 3 Bits der Pixelreihe eingetragen haben.
Wir haben jetzt alle Bits der Y-Koordinate eingesetzt, bei der Bildadresse sind aber noch 3 unbenutzte Bits ganz oben im High-Byte übrig. Mit ihnen kann man noch die Startadresse des Bildspeichers festlegen – das ist normal „010" (binär) für einen Start ab Adresse 16384 = 4000 hex. Man kann aber auch „100" (binär) für einen Bildspeicher, der ab 32768 = 8000 hex im Speicher liegt (auf dem 48k und 128k Speccy) verwenden, wenn man z.B. unsichtbar ein zweites Bild aufbauen will oder um beim 128k Spectrum auf den (über eine Speicherseite eingeblendeten) zweiten Bildspeicher zuzugreifen.

Hier das Assembler-Programm für die Berechnung der kompletten Bildadresse:

☛ Info zu den Befehlen „CP" und „JP": Das C(arry)-Flag wird gesetzt, wenn der Wert im Register A kleiner als neben „CP" ist.
☛ Der „AND"-Befehl löscht das Carry-Flag, deswegen können wir unten „RLA" statt „SLA A" verwenden und ein wenig Rechenzeit sparen.
☛ Wenn ihr andere Sprite-Maße haben wollt, ab „Getattr" Zeile 2, 4, 6 und 8 ändern !

Sprite-Maße im Beispiel: 4*20 Bytes

Register-Eingabe: B = x-pos. (0...31)
C = y-pos. (0...191)

Register-Ausgabe:
DE = Sprite-Anfangsadresse im Bildspeicher

```
GETADDR LD  A,B      ; Test: x zu gross ?
        CP  29       ; (= 33 - Spritebreite)
        JP  C,VALXOK
        LD  B,28     ; Korrigiere x
VALXOK  LD  A,C      ; Test: y zu gross ?
        CP  173      ; (= 193 - Spritehoehe)
        JP  C, VALYOK
        LD  C,172    ; Korrigiere y
VALYOK  LD  E,B      ; x-pos. eintragen
        LD  A,C      ; y-pos. (Pix.Reihe)
        AND 7        ; (untere 3 Bits)
        LD  D,A
        LD  A,C      ; y-pos. (Zeile)
        AND 56       ; (mittlere 3 Bits)
        RLA          ; (um 2 Bits hoch)
        RLA
        OR  E        ; (zu x-pos. dazu)
        LD  E,A
        LD  A,C      ; y-pos. (Block)
        AND 192      ; (obere 2 Bits)
        RRA          ; (um 3 Bits runter)
        RRA
        RRA
        OR  D        ; (zu y-pos. Reihe)
        OR  64       ; Screen Anfang dazu
        LD  D,A
        RET
```

3.2. Eine einfache übermalende Sprite-Routine

Diese Sprite-Routine ist sehr einfach und kurz, übermalt aber den Bildspeicher ohne Rücksicht auf seinen augenblicklichen Inhalt. Ein Männchen als Sprite sieht dadurch so aus, als ob es sich in einem Bilderrahmen befindet, der herumgeschoben wird (der Hintergrund wird verdeckt!). Wenn dieser „Bilderrahmen" auch am Rand gesetzte Pixel besitzt, hinterläßt er darüber hinaus bei Bewegungen eine Spur – wenn das nicht erwünscht ist, sollte man bei der Sprite-Grafik ringsum ein Byte vom Rand her gesehen leer (auf 0) lassen.

Zum Ablauf der Routine:

Die Sprite-Anfangsadresse im Bildspeicher wird durch einen Aufruf von unserer obigen Routine „GETADDR" berechnet und in dem Registerpaar DE gespeichert. Eine X-Schleife (Sprite-Breiten-Schleife) gibt es in dieser Sprite-Routine nicht, das erledigt der „LDIR" (= kopiere 1 Byte von (HL) nach (DE) und zähle BC herunter bis es auf 0 ist). B enthält die Sprite-Höhe, muß aber vorübergehend für den LDIR auf 0 gesetzt werden (der Inhalt von B wird deswegen nach A gerettet).

Nach dem LDIR werden die 4 davon in X gemachten Schritte wieder zurückgenommen (mit Beachtung des Übertrags von E nach D!). Danach geht es an die Berechnung der Bildadresse der nächst-tieferen Pixelreihe (s.h. oben, die erwähnte 2. Berechnungs-Variante kommt jetzt!). Falls dabei ein Überlauf entsteht, erhöht dies automatisch die Nummer des Blocks (s.h. High-Byte auf Abbild oben). Das korrigieren wir hier erst einmal nicht, es kann ja sein, daß wir die Blocknummer sowieso noch erhöhen müssen – statt dessen erhöhen wir zunächst die Zeilennummer. Erst wenn es dabei keinen Überlauf gibt, wissen wir, daß der Block-Überlauf falsch war und korrigieren ihn (der Zeilen-Überlauf muß nicht korrigiert werden, weil er im Carry-Flag landet und dieses wird nicht weiter beachtet). Damit ist die Berechnung der nächsten Bildadresse schon abgeschlossen. Mit einem „DJNZ" (zähle B herunter und springe, solange B nicht 0 ist) schließen wir die Y-Schleife (Sprite-Höhen-Schleife) ab.

☛ Wenn ihr andere Sprite-Maße haben wollt, Routine „Getattr" und Zeilen 2, 3, 9 ändern !

Sprite-Maße im Beispiel: 4*20 Bytes

Register-Eingabe: HL = Adresse Sprite-Grafik-Daten
B = x-pos. (0...31),C = y-pos. (0...191)

```
SPRITE_1 CALL GETADDR  ; Sprite Startadr. -> DE
         LD   B,20     ; Sprite-Hoehe
LOOP     LD   C,4      ; Sprite-Breite
         LD   A,B      ; B retten
         LD   B,0
         LDIR          ; 1 Pixelreihe setzen
         LD   B,A      ; B wiederherstellen
         LD   A,E      ; Zurueck auf x-Anfang
         SUB  4        ; (Sprite-Breite)
         LD   E,A
         JR   C,CONT1  ; Ueberlauf -> D nicht
                       ; aendern
         INC  D        ; Naechste Pixelreihe
CONT1    LD   A,D      ; Überlauf (= Block+1) ?
         AND  7
         JP   NZ,CONT2 ; Nein -> weiter
         LD   A,E      ; Naechste Zeile
         ADD  A,32
         LD   E,A
         JR   C,CONT2  ; Ueberlauf -> weiter
         LD   A,D      ; Ueberlauf Reihe-
                       ; >Block
         SUB  8        ; korrigieren - kein
         LD   D,A      ; neuer Block !
CONT2    DJNZ LOOP     ; Ende Hoehen-
                       ; Schleife
         RET
```

☛ Um damit jetzt einen Sprite darstellen zu können, braucht ihr zuerst einmal Sprite-Grafikdaten. Sie können ganz einfach linear (Byte für Byte, Zeile für Zeile) im Speicher abgelegt werden, d.h. ihr müßt sie nicht so kompliziert wie im Bildspeicher anordnen. Das einfachste ist aber vielleicht, wenn ihr eine 32*20 Pixel große Figur mit einem Zeichenprogramm in die linke obere Ecke des Bildschirms malt und sie dann mit einer umgekehrten Sprite-Routine einlest (sie liest vom Bild und schreibt in den Speicher): Ihr müßt nur direkt vor und hinter dem „LDIR" ein „EX DE,HL" einfügen und schon habt ihr eine Sprite-Einlese-Routine !

☛ Falls euch beim Bewegen des Sprites ein Flimmern auffällt: Das kommt dadurch, daß der Computer beim Zeichnen des Sprites vom Elektronenstrahl des Bildschirms überholt wird und so die eine Hälfte des Sprites noch mit aufs Bild kommt, während die andere Hälfte erst 1/50 Sekunde später folgt. Falls ihr also einen Sprite (oder mehrere schnell hintereinander) setzen wollt, solltet ihr vorher einen HALT-Befehl verwenden (die Interrupts sollten dabei aber eingeschaltet sein). Dadurch wartet der Computer, bis der Elektronenstrahl ganz links oben auf dem Bildschirm angekommen ist, bevor er anfängt, die Sprites zu zeichnen. Meist hat der Computer dann genug Zeit, bis der Elektronenstrahl ihn einholt.

3.3. Modifikation zu einer invertierenden Sprite-Routine

Der Vorteil von einer invertierenden Sprite-Routine ist, daß die Sprite-Grafik den Bild-

Hintergrund nicht zerstört – statt dessen wird sie mit ihm zusammengerechnet (verknüpft), das kann aber zu unschönen Veränderungen in der Grafik führen. Wenn man den Hintergrund später wiederherstellen will, braucht man den Sprite einfach nur noch einmal auf die gleiche Stelle zu setzen, denn durch eine weitere Verknüpfung wird die Sprite-Grafik wieder vom Hintergrund getrennt (auseinandergerechnet) und verschwindet restlos.

Wenn man einen Sprite bewegen will, sieht der Ablauf also wie folgt aus:

1. Sprite setzen
2. Neue Sprite-Position berechnen
3. Sprite an alter Position noch einmal setzen, um dort den Hintergrund wiederherzustellen
4. Sprite an neuer Position setzen
5. weiter bei 2.

☛ Achtung, bei der neuen Sprite-Routine sind die Register B und C (enthalten Breiten- und Höhe-Zähler) gegenüber der ersten Routine vertauscht !

☛ Alle Zeilen mit einem „*" im Kommentar haben sich geändert.

☛ Wenn ihr andere Sprite-Maße haben wollt, Routine „Getattr" und Zeilen 2, 3, 11 ändern !

Sprite-Maße im Beispiel: 4*20 Bytes

Register-Eingabe: HL = Adresse Sprite-Grafik-Daten
B = x-pos. (0...31),C = y-pos. (0...191)

```
SPRITE_2 CALL GETADDR ; Sprite Startadr. -> DE
         LD   C,20    ; * Sprite-Hoehe
LOOP1    LD   B,4     ; * Sprite-Breite
LOOP2    LD   A,(DE)  ; * 1 Screen-Byte lesen
         XOR  (HL)    ; * mit Daten ver
                      ; knuepfen
         LD   (DE),A  ; * Screen-Byte
                      ; schreiben
         INC  HL      ; * Datenzeiger+1
         INC  DE      ; * Screenzeiger+1
         DJNZ LOOP2   ; * Ende Breiten-
                      ; Schleife
         LD   A,E     ; Zurueck auf x-Anfang
         SUB  4       ; (Sprite-Breite)
         LD   E,A
         JR   C,CONT1 ; Ueberlauf -> D nicht
                      ; aendern
         INC  D       ; Naechste Pixelreihe
CONT1    LD   A,D     ; Überlauf (= Block+1)?
         AND  7
         JP   NZ,CONT2; Nein -> weiter
         LD   A,E     ; Naechste Zeile
         ADD  A,32
         LD   E,A
         JR   C,CONT2 ; Ueberlauf -> weiter
         LD   A,D     ; Ueberlauf Reihe
                      ; ->Block
         SUB  8       ; jetzt korrigieren - kein
         LD   D,A     ; neuer Block !
CONT2    DEC  C       ; * Reihenzaehler – 1
         JP   NZ,LOOP1 ; * Ende Hoehen-
                      ; Schleife
         RET
```

3.4. Eine Sprite-Routine mit Maske

Eine gute Sprite-Routine arbeitet mit einer Maske – sie verhindert, daß um den Sprite herum ein leerer, den Hintergrund übermalender Rahmen (wie bei unserer ersten Routine) ist und es gibt auch keine Grafikfehler (wie bei unserer zweiten Routine). Der Sprite kann dank Maske jetzt dreierlei Typen von Pixeln darstellen: Durchsichtig (d.h. Hintergrund zu sehen), gelöscht (Pixel-Bit auf "0") und gesetzt (Bit auf "1"). Bei dem invertierenden Sprite konnte man z.B. durch das Weiße im Auge einer Spielfigur den Hintergrund sehen – das kann jetzt durch die Maske verhindert werden.

Mit einer Maske „schneidet" man den Teil aus dem Bild-Hintergrund heraus (d.h. setzt alle seine Pixel auf „0"), der später durch die Sprite-Grafik belegt werden soll. Die Maske hat die gleichen Maße wie die Sprite-Grafik und besitzt an allen Stellen, wo der Sprite durchsichtig sein soll, gesetzte Bits („1"), der Rest ist auf „0". Der Ablauf um ein Sprite-Byte mit Maske im Bildspeicher zu setzen, sieht wie folgt aus:

1. Ein Byte aus dem Bild lesen.
2. Dieses Bild-Byte mit Masken-Byte „AND"-verknüpfen (dadurch wird

Platz für die Sprite-Pixel freigemacht)
3. Das Ergebnis mit Sprite-Byte „OR"-verknüpfen (Sprite-Pixel werden hinzugefügt)
4. Das daraus resultierende Byte wieder in den Bildspeicher kopieren

Diese Sprite-Routine mit Maske verändert den Bildhintergrund (im Gegensatz zur invertierenden Routine oben) unwiederbringlich, daß heißt, daß wir den Hintergrund vor dem Setzen der Sprite-Grafik retten müssen, damit wir ihn später wiederherstellen können. Der Ablauf für eine Sprite-Bewegung von einer Position zur nächsten sieht dann wie folgt aus:

1. An der Position, wo der Sprite erscheinen soll, erst mal den Bild-Hintergrund einlesen (sichern)
2. Hintergrund mit Maske (in Sprite-Größe) AND-verknüpfen (es wird Platz für den Sprite gemacht)
3. Hintergrund mit der Sprite-Grafik OR-verknüpfen (Sprite erscheint)
4. Neue Position für Sprite berechnen
5. An alter Position Sprite wieder durch den bei 1. gesicherten Hintergrund übermalen (mit erster übermalender Sprite-Routine) – Sprite verschwindet
6. Weiter bei 1., aber an neuer Position !

Hier noch ein Tip, wenn ihr mehrere Sprites darstellen/bewegen wollt: Die Bild-Hintergründe der Sprites dürfen nur dann eingelesen werden, wenn kein Sprite sichtbar ist ! Also erst alle Hintergründe einlesen, dann alle Sprites auf einmal zeichnen und für die nächste Bewegung erst mal alle Hintergründe wiederherstellen !

Nun zur Routine:

Die Erweiterung von der invertierenden zu einer maskierten Sprite-Routine ist sehr einfach, es sind nur ein paar kleine Änderungen nötig (siehe unten, wieder mit „*" markiert). Anstatt erst die komplette Maske und dann den kompletten Sprite mit dem Bild-Hintergrund zu verknüpfen, machen wir das hier gleichzeitig bei jedem Bild-Byte und ersparen uns dadurch Rechenzeit (nur ein Durchlauf durch die Sprite-Schleifen). Die Maske muß jetzt zusammen mit der Sprite-Grafik gespeichert werden (im Wechsel ein Byte Maske, ein Byte Sprite-Grafik), damit sie schnell erreichbar ist ! Dadurch kann man die obige Sprite-Einlese-Routine jetzt nicht mehr verwenden, um die Sprite-Grafik aus dem Bildschirm einzulesen ! Beim Setzen des Sprites in den Bildspeicher wird nun immer erst das Masken-Byte aus dem Sprite-Grafikspeicher gelesen und mit dem Bild-Byte per AND verknüpft. Anschließend wird dann das Sprite-Grafik-Byte gelesen und per OR mit dem Bild-Byte verknüpft. Die Bildadresse bleibt dabei gleich.

Für den kompletten Ablauf (Sprite setzen und wieder entfernen) brauchen wir außer der folgenden Routine noch die übermalende und die normale Sprite-Einlese-Routine, um den Hintergrund zu retten und nachher wieder herzustellen.

☛ Wenn ihr andere Sprite-Maße haben wollt, Routine „Getattr" und Zeilen 2, 3, 13 ändern!

Sprite-Maße im Beispiel: 4*20 Bytes

Register-Eingabe: HL = Adresse Sprite-Grafik-Daten
B = x-pos. (0...31), C = y-pos. (0...191)

```
SPRITE_3  CALL GETADDR  ; Sprite Startadr. -> DE
          LD   C,20     ; Sprite-Hoehe
LOOP1     LD   B,4      ; Sprite-Breite
LOOP2     LD   A,(DE)   ; 1 Screen-Byte lesen
          AND  (HL)     ; * mit Maske ver-
                        ; knüpfen
          INC  HL       ; * weiter zu Daten
          OR   (HL)     ; * mit Daten ver-
                        ; knüpfen
          LD   (DE),A   ; Screen-Byte schreiben
          INC  HL       ; Datenzeiger+1
          INC  DE       ; Screenzeiger+1
          DJNZ LOOP2    ; Ende Breiten-
                        ; Schleife
          LD   A,E      ; Zurueck auf x-
```

```
                                ; Anfang
         SUB 4                  ; (Sprite-Breite)
         LD  E,A
         JR  C,CONT1            ; Ueberlauf -> D nicht
                                ; aendern
         INC D                  ; Naechste Pixelreihe
CONT1    LD  A,D                ; Überlauf (= Block+1)?
         AND 7
         JP  NZ,CONT2           ; Nein -> weiter
         LD  A,E                ; Naechste Zeile
         ADD A,32
         LD  E,A
         JR  C,CONT2            ; Ueberlauf -> weiter
         LD  A,D                ; Ueberlauf Reihe-
                                ; >Block
         SUB 8                  ; korrigieren - kein
         LD  D,A                ; neuer Block !
CONT2    DEC C                  ; Reihenzaehler – 1
         JP  NZ,LOOP1           ; Ende Hoehen-
                                ; Schleife
         RET
```

3.5. Fortgeschrittene Programmierung: Eine sehr schnelle übermalende Sprite-Routine

Wenn es sehr auf Geschwindigkeit ankommt, muß man schon mal tiefer in die Trickkiste greifen. Die häufigsten Zeit-Killer sind Berechnungen – man sollte wenn irgendwie möglich auf sie verzichten, das heißt, viel mit Tabellen arbeiten, die vorberechnete Werte enthalten. Außerdem sollte man versuchen, so viele Werte wie möglich in den Prozessor-Registern zu halten, anstatt sie jedes mal aus dem Speicher zu lesen. Komplexe Befehle (wie „LDI", „LDIR", „PUSH", „POP" oder „DJNZ") können auch einige Zeit sparen. Eine weitere Möglichkeit ist auch, bei Sprung-Stellen, wo die Bedingung eher selten zutrifft, den „JR"-Befehl statt dem „JP" einzusetzen, weil „JR" weniger Zeit benötigt, wenn nicht gesprungen werden muß. Das kann sich vor allem in häufig ausgeführten Programmteilen (wie Schleifen) lohnen. Der Nachteil von „JR" ist, daß sich die Zeit, die eine Routine zum Ablauf benötigt, schwerer berechnen lässt – die genaue Zeit braucht man z.B. bei der Darstellung von Farbbalken im Border oder beim Speichern auf Kassette.

Als ein Beispiel für Geschwindigkeits-Optimierung möchte ich hier die erste Sprite-Routine noch einmal in geänderter Form vorstellen. Die Adresse der jeweils nächsten Pixelreihe wird jetzt nicht mehr berechnet, sondern aus einer Tabelle entnommen – das spart ungemein viel Zeit (wie schon an der Kürze der Hauptschleife der Routine unten zu erkennen ist). Weil im Speicher aber kein Platz für alle Bildadressen ist, wird hier nur die Adresse des ersten Bytes jeder Pixelreihe in der Tabelle abgelegt. Um dann an die tatsächliche Bildadresse zu kommen, wird die X-Position noch aufaddiert (das ist zwar eine Berechnung, sie kostet aber nicht viel Zeit). Noch ein weiterer unter Demo-Programmierern gern genutzter Trick: Auf die Tabelle wird per Stack-Pointer (Register „SP") zugegriffen, weil das die schnellste Methode ist - ein „POP" reicht, um einen 16-Bit-Wert auszulesen, schneller geht's nicht ! Der Stack wird also zu einem Tabellen-Zeiger umfunktioniert, mit dem Nachteil, daß die Interrupts während des Ablaufs der Routine gesperrt werden müssen, um einen Absturz zu verhindern ! Während der Zeit können auch keine Unterprogramme (CALL, RET) aufgerufen werden.

Vor der ersten Benutzung der Routine muß man die Tabelle mit den Bildadressen erzeugen, dazu „MAKE_TAB" (weiter unten abgedruckt) aufrufen ! Das ist nur einmal notwendig.

Grober Ablauf der Sprite-Routine:

Weil kein Register mehr im Hauptregistersatz frei war, verwendet die Routine selbstmodifizierenden Code, um die X-Position am Anfang der Routine direkt in den „LD A,0"-Befehl aus der Sprite-Hauptschleife zu speichern. Dieser LD-Befehl ist zwei Bytes lang, wobei das erste Byte den Befehlscode darstellt und das zweite die Zahl, die in das A-Register geladen werden soll - man braucht also nur das zweite Byte mit dem neuen Wert zu überschreiben. Als nächstes wird die

angegebene Y-Position in einen Zeiger auf die Bildadressen-Tabelle umgerechnet und dann im Register SP abgelegt. Für die Hauptschleife wird das B-Register als Zeilen-Zähler verwendet. Per Stack-Pointer wird der Beginn der nächsten Pixel-Reihe aus der Tabelle gelesen und es wird die X-Position hinzuaddiert. Die vier „LDI"s kopieren dann die Sprite-Daten (4 Bytes) ins Bild. Nach 20 Schleifen-durchläufen muß der Stack-Pointer wiederhergestellt werden- und das bevor man die Interrupts wieder einschaltet.

☛ Der Stack-Pointer wird hier als Zeiger auf eine Tabelle mit allen Pixelreihen-Anfangs-adressen des Bildschirms verwendet.

☛ Wenn ihr andere Sprite-Maße haben wollt, Zeile 13 und die Zahl der „LDI"s ändern !

Sprite-Maße im Beispiel: 4*20 Bytes

Register-Eingabe: HL = Adresse Sprite-Grafik-Daten
B = x-pos. (0...31), C = y-pos. (0...191)

```
SPRITE_4  DI                    ; Interrupts sperren
          LD    A,B             ; X-Offset
                                ; speichern
          LD    (X_POS+1),A
          LD    (SP_SAVE),SP; SP retten
          EX    DE,HL           ; HL retten
          LD    H,0             ; Y-Pos * 2
          LD    L,C
          ADD   HL,HL
          LD    BC,SP_TAB       ; Tab-Start dazu
          ADD   HL,BC
          LD    SP,HL           ; in SP speichern
          EX    DE,HL           ; HL wiederher-
                                ; stellen
          LD    B,20            ; Sprite-Hoehe
LOOP      POP   DE              ; Naechste Screen-
                                ; Zeile
X_POS     LD    A,0             ; X-Pos dazu-
                                ; addieren
          ADD A,E
          LD    E,A
          LD    A,B             ; B retten
          LDI                   ; 4 Bytes setzen
          LDI
          LDI
          LDI
          LD    B,A             ; B wiederher-
                                ; stellen
          DJNZ  LOOP            ; Ende Hoehen-
                                ; Schleife
          LD    SP,(SP_SAVE)    ; SP wiederher-
                                ; stellen
          EI                    ; Interrupts
                                ; einschalten
          RET

; — Screen-Adressen-Tabelle erzeugen —

MAKE_TAB LD     HL,SP_TAB ; Zeiger: Anfang der
Tab.
          LD    B,192           ; 192 Zeilen
          LD    DE,16384        ; 1. Screen-Adresse
MK_LOOP   LD    A,E             ; Adresse in Tab.
                                ; Speichern
          LD    (HL),A
          INC   HL
          LD    A,D
          LD    (HL),A
          INC   HL
          INC   D               ; Naechste Screen-
                                ; Adresse
          LD    A,D
          AND   7
          JP    NZ,MK_CONT
          LD    A,E
          ADD   A,32
          LD    E,A
          JR    C,MK_CONT
          LD    A,D
          SUB   8
          LD    D,A
MK_CONT   DJNZ  MK_LOOP
          RET
SP_TAB    DEFS 384              ; Pixelreihen-
                                ; Adressen
SP_SAVE   DEFW0                 ; Original-Stack-
                                ; pointer
```

So, hier ist Schluß. Ich hoffe, daß jemand meinen Artikel bis hierhin gelesen hat, daß meine Erklärungen halbwegs verständlich waren und daß ihr vielleicht Lust bekommen habt, mal ein kleines Spielchen mit den Sprite-Routinen zu schreiben (für das Spiel selbst reicht ja bereits Basic).

Tschau!

Christof Odenthal
c_odenthal@web.de



Wave-files via Quazar

At the moment I am looking to minimize my SAM surroundings. My „big box" (something like a PC-Tower) which contains some interfaces becomes more and more useless since I have an internal Atom-Interface and my new mouse. The box also contains my first harddisk with external interface and an Eddac-Soundinterface (which was my first one). The only use I had for it was playing Wave-files, as Martijn Groens Waveplayer can only handle the Eddac.
I have buyed the Quazar Surround Mixer Lead cable from Colin. One end of the cable will be plugged into SAMs Lightpen socket and and the other into the front speaker socket on the Quazar. The signals from SAMs soundchip (in stereo) will now played via Quazar and my extended loudspeakers. So I come to the idea to ask Colin, if there is a way to play Wave-files also via his Quazar. As usual I did it via email:

Wo: Is it also possible to get the waves played via your Quazar? If you have B-Dos higher 1.7 then you should have Martijns Waveplayer. It works well, but only with an Eddac. However, if I could get run them with yours, then I should have all sorts of music/tunes then via Quazar.

Colin: There has never been the need for a seperate program to play Waves .... the utilities you need to convert Waves into proper Quazar Surround samples are on the 'Introductory Disk 2' that came with the Quazar Surround.

As it is very easy to convert Waves into the sample format for the Quazar Surround - why use ugly PC file type when I make special sample types for playing on the Quazar Surround!

Here is a very quick run through of how to convert waves:

First, remove the header at the start of the Wave file ..... this is very crude, but does the job nicely!

LOAD "sample.wav" CODE 32768
SAVE "sampledata" CODE 32768+64, filelength-64

Now we have a file with just the sample data....

Next use the 'Sample Convertor" program on "Intro disk 2" to prepare this 'sampledata' file - use options 3,4,5 depending on what type of sample it was,and then let the program save off the converted data.

Last step is load the saved data from the Sample Convertor into the "Rate Converter" on "Intro disk 2". This program will reduce the sample rate of the file to allow it to play with the normal Quazar Surround sample players.... select convert to 7.8KHz if the wave file was 11025, or 15.6KHz if the wav file was 22050 or above...

Once that program has done it's conversion the sample is ready for playing with the Quazar Surround Sample Players, or for use in any other program that uses Quazar Surround samples .... such as Quazar Studio and Quazar Sequencer Pro.....

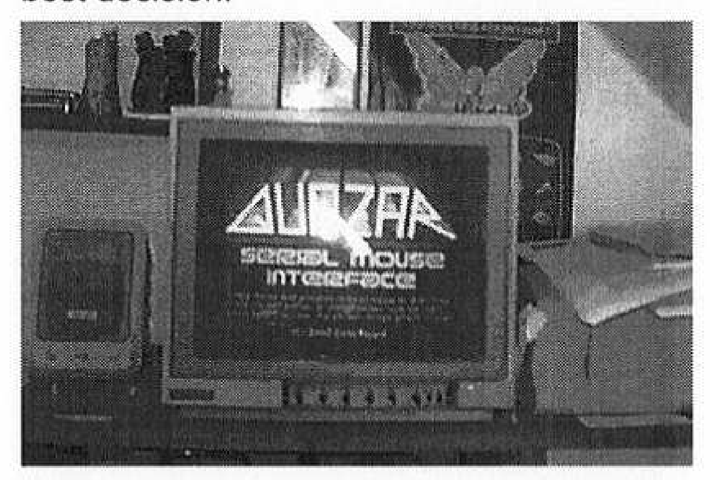
Once you get used to using the converter programs it is very easy :)

Thank you Colin. I will try it out and then I write down my experiences.

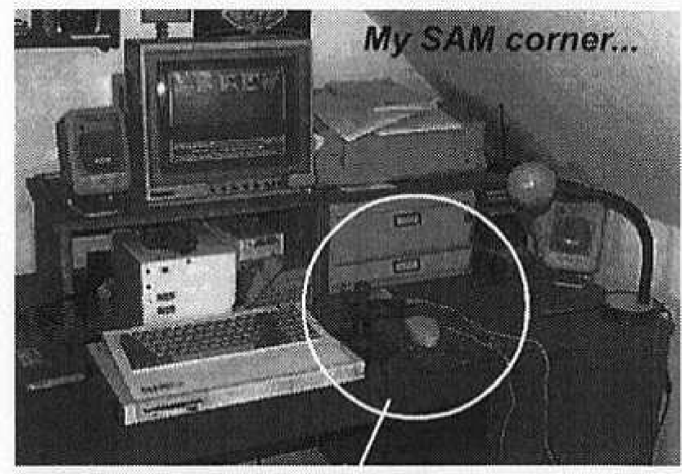
And since I am the happy winner of a pair of wireless funk controlled loudspeakers (by a german brewery) I will start to push my SAM to a real musicbox :-)))))) *(Wo)*

From man and mices..

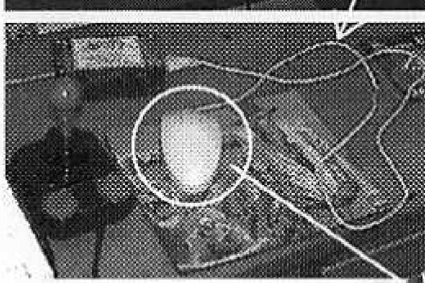
It seems years ago that I got my original SAMCO mouse interface and it seems also years ago, that I could never used it really. The reason was that I never got a smooth running mouse. I tried out several mices with no success, the cursor „jumped" inaccurat. But when I was on a meeting and used the mouse from Dirk or Robert, they run well. Both use Atari mices, but the two I got didn't their job. No wonder, that I had given up.
Long time it was an idle wish to play Lemmings or Las Vegas (my favorite cardgame) like on a Mac or PC.
The solution came with Colins mouse interface advert. I decided to buy them, it was my best decision.



The interface came one day packed with a mouse and a disk. On the disk there is a simple program to make you familiar with the mouse handling. Pressing the left buton will change the colour, pressing the right one will clear the screen. You can not really paint with this program, but for such things we have Flash or Sampaint, both programs didn't bring fun and makes sense without a mouse. So Sampaint was the first I tried out.
What's to say. I was filled with enthusiasm. The mouse runs smooth over the screen, sets exactly the points or lines I want and there was no problem to open or close windows, i.e. for choosing a Record.
Encouraged by this I started to play Lemmings. And what's to say? You can quickly



My SAM corner...



The new interface with mouse.

na, it's a joke of course!

choose the required figure to play proper through all the levels. Great.
Not to mention that I tried also some sorts of cardgames that supports a mouse. As described, I got no problem with any sort of a mousedriver.
Now I have to learn how to write programs with mouse control. First steps are:

POKE SVAR 409,(1=mousedriver enabied, 0=disabled)
POKE SVAR 666,(0=mousearrow invisible, 1-8=one of 8 mouse icons)
POKE SVAR 406,x (x=sets x-coord)
POKE SVAR 408,y (y=sets y-coord)

On the last Bunnik meeting I showed Martijn and Robert my recent acquisition. Robert, who is an excellent artist does a test and I can write here down his compliments to Colins work.
Now I am looking for all sorts of programs which supports mouse steering and I will make a list of them. Unfortunately the World Cup has stopped my action for a while, but I think, you can read about this list in the next magazine. *(Wo)*

Tipshop: Boing!

(Noesis, Nov 1992)

* Lift key found in screen More Of The Hill and needed for lifts in two places.
* Parachute found in screen More Spooky Floaty Things and needed to drop into shaft in The Garden.
* Pickaxe found in screen Summer House and needed to open shaft in The Garden.
* Goggles found in screen The Very Top Of The Hill and needed to enter I'm Blind Room.
* Spade found in screen Ouch and needed to open Burial Mound.
* Dynamite found in screen The Master Bedroom and needed, with plunger, in Dead End Room.
* Plunger found in screen Splat (Or, The Bottom) and needed, with dynamite, in Dead End Room.
* Barrel one found In screen The Entrance Hall and needed to cross pit in caves.
* Barrel two found in screen Even More Floaty Things and needed to cross the pit in the caves.
* Hearts found in six different screens and restore air supply to full.
* Potion bottles found in twenty different places and you need to collect them all.

TIME MACHINE PARTS

* Crystal - in the caves.
* Switch - In teleport room above I'm Blind Room.
* Battery - In Deepest Chamber at the back of the caves.



TELEPORTERS

* In Beam Me Up room - Use left to go to Front Door, and right to go to Bridge To Nowhere.
* In The Teleporter room - Use to go to Summer House,
* In The Crypt - Use to go to Strange. After collecting all three parts of the time machine and all twenty potions, return to start and the time machine will appear. Enter this and you have finished the game.

  In the screen titled Spooky Floaty Things 3, you have to jump off the third platform from the top to land on a platform in the next screen.

(Patrick Spencer, Tony Baitson in Your Sinclair Issue 89)

Tipshop: Splat!

(Revelation, Apr 1992)

When the loading screen pops up press and then type the folowing...

35 LET lives=*,lf=1, level=1 ,lev=*, score=* (instead of stars, put numbers)

Now run the program and you will have as many lives as you selected and will also be on the level you selected. That wasn't too painful, was it?

(Christopher Bailey in Your Sinclair issue 79)



Found on: ***http://www.the-tipshop.co.uk/***

Interscript

Hi Freaks!!

Wenn man, so wie wir, mal wieder in der Routinenkiste kramt, stößt man des öfteren auf ganz nützliche Programme und ein solches wollen wir uns heute einmal ansehen. Es heißt INTERSCRIPT und wurde von Dirk Zweers geschrieben. INTERSCRIPT ist eine interruptgesteuerte Laufschriftroutine, die es ermöglicht, während der Abarbeitung eines Programmes Informationstext auf dem Bildschirm auszugeben. INTERSCRIPT hat eine ganze Reihe von Vorteilen aufzuweisen, als da wären:

Freie Verschiebbarkeit im Speicher (läuft allerdings nur auf Spectrum 48K, da im oberen Speicherbereich zwischen 65524 und 65535 einige Variablen abgelegt werden).

Positionierung und Breite der Laufschrift (Windowpositionen) sind frei definierbar.

Die im Speicher festgelegten Texte sind von ihren Adressen her nicht an bestimmte Werte gebunden.

Die Routine beeinträchtigt ablaufende Programme kaum, da sie im Interrupt abläuft wodurch auch die Scroll-Routine nicht in Mitleidenschaft gezogen wird.

Die für solche Programmabläufe benötigte Interrupt-Tabelle wird hier nicht gebraucht, da das Programm sehr eng sowohl soft- als auch harwaremäßig an den Spectrum angepaßt wurde.

Ständige Kontrolle des gerade ausgedruckten Textes mit der Möglichkeit jederzeit auf einen anderen Text umzuschalten.

Die Routine springt beim Erreichen des Textendes wieder an den Anfang.

Die Routine kann jederzeit abgeschaltet werden.

Wenn man diese Punkte näher betrachtet, so sieht man, daß doch einiges in dieser kleinen Routine steckt. Das am Ende folgende BASIC-Listing erzeugt den notwendigen Maschinencode und beinhaltet eine kleine Demotextversion. Wenn der MC-Teil z. B. an der Adresse x abgelegt wurde, muß man die PRINT-Positionen festlegen. Das sieht dann beispielsweise so aus:

```
RANDOMIZE USR X: PRINT 0,7,13
```

Das bedeutet folgendes. Die Laufschrift erscheint auf dem Bildschirm ab PRINT-Position 0,7 mit einer Breite von 13 Zeichen. Die erste Zahl (im Beispiel 0) darf im Bereich 0- 23 liegen, die zweite im Bereich 0- 31 und die dritte im Bereich 1- 32. Das ist auch ganz logisch, wenn man die Bildschirmparameter beachtet. Ist einer der Werte für die zweite oder dritte Zahl nicht innerhalb dieser Grenzen, dann kommt es zu der Fehlermeldung A - Invalid Argument. Wenn man einen Text erstellt und in den Speicher gepoket hat, z. B. ab Adresse 32000, so übergibt man diesen Wert folgendermaßen an das Programm:

```
RANDOMIZE USR 64002: PRINT 32000
```

Wobei 64000 ein idealer Platz für die Abspeicherung der Routine ist. Damit richtet man den Textzeiger auf den auszugebenden Text. Das Ende muß man mit CHR$ 255 (COPY) definieren. Das läßt sich ganz einfach dadurch bewerkstelligen, wenn man an den Text, der etwa in x$ abgelegt ist, folgende Zeile anfügt:

```
LET x$=x$+CHR$ 255
```

Um einen Text in den Speicher zu poken. kann man folgende kleine aber wirkungsvolle Routine benutzen:

```
FOR a=1 TO LEN x$: POKE a+(b-1),
x$(a): NEXT a
```

In x$ muß dabei natürlich der entsprechende Text stehen und b ist die Adresse, ab der der Text abgelegt werden soll. In diesem Zusam-

menhang muß noch ganz besonders darauf hingewiesen werden, daß die Routine keine TOKENS ausdruckt und auf die Zeichensatzvariable CHARS zurückgreift. Damit ist es möglich, eigene Zeichensätze zu verwenden. Es werden dann bis zu 255 Zeichen (255 ist ja für die Endkennung reserviert) ausgegeben. Der einzupokende Text kann natürlich in Abhängigkeit von der Speichergröße beliebig lang sein. Wenn schließlich alle Werte initialisiert sind kann man die Routine mit RANDOMIZE USR 64004 starten. Damit wird der Interrupt aktiviert und es geht los. Abschalten kann man das alles wieder mit RANDOMIZE USR 64006. Um letztendlich noch festzustellen, an welcher Textposition die Routine gerade ist (interessant insbesondere bei langen auszugebenden Texten), kann man den folgenden Befehl benutzen:

```
LET z=USR 64000
```

Z (oder irgendeine andere von uns definierte Variable) beinhaltet dann die Adresse des Buchstabens, der als nächster gedruckt wird.

Soviel also zur Bedienung der Routine "INTERSCRIPT". Und jetzt noch das Listing zum Abtippen:

```
 1 INPUT '.STARTADRESSE ? .' ; A
 2 FOR B=1 TO 25
 3 RESTORE (B+20)
 4 LET C=0
 5 FOR D=1 TO 10
 6 READ F
 7 POKE A.F
 8 LET C=C+F
 9 LET A=A+1
10 NEXT D
11 READ G
12 IF C<>G THEN PRINT FLASH 1;
   "..FEHLER IN DATAZEILE ";B+
   20'"..GELESENE PRUEFSUMME: ",
   G'"ERRECHNETER WERT: ";C:
   BEEP .1,0: STOP
13 PRINT AT 0,0;"DATAZEILE "; B+
   20;" O.K.": BEEP .1,10
14 NEXT B
15 GO TO 9999
21 DATA 24,49,24,119,24,7,24,38,
   237,75,621
22 DATA 251,255,201,243,171,137,
   0,197,225,25,1551
23 DATA 34,245,255,62,195,50,
   244,255,62,24,1426
24 DATA 50,255,255,62,59,237,71,
   62,8,50,1109
25 DATA 254,255,237,94,251,201,
   243,237,70,251,2093
26 DATA 201,223,231,62,3,245,
   231,205,130,28,1559
27 DATA 205,153,30,241,197,61,
   32,243,193,121,1476
28 DATA 167,40,21,111,254,33,48,
   16,193,89,972
```

Vielleicht kann sie ja der eine oder andere von euch für ein Programm gebrauchen. Jedenfalls wünschen wir euch bein Ausprobieren viel Spaß damit. Bis bald hier im Info.

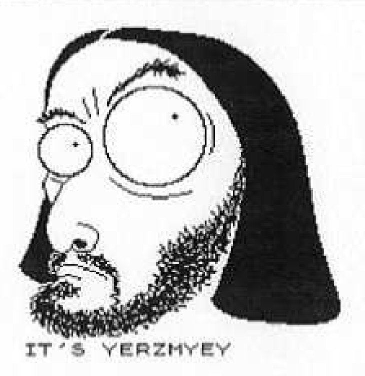
Harald R. Lack. Heidenauer Str. 5
83064 Raubling
Hubert Kracher, Schulweg 6
83064 Großholzhausen

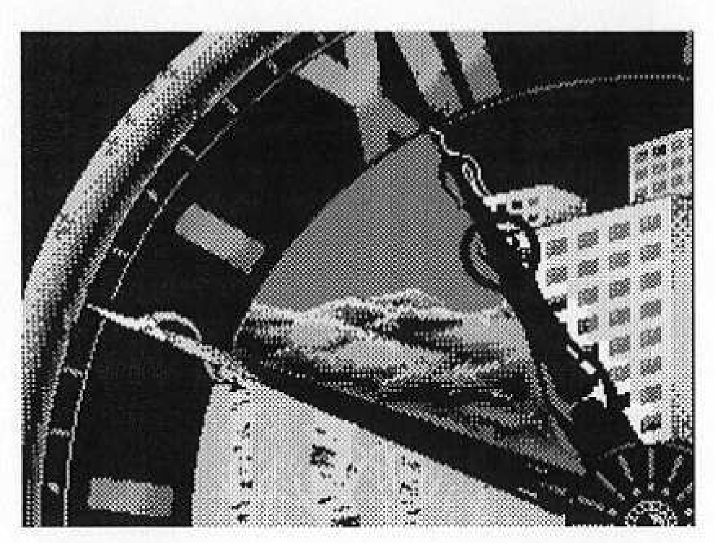
Hut ab!

Letztesmal berichtete ich von einer Meldung in „Mumpitz“ zu Heinz Schobers Artikel über angebliche Spectrumviren in unserem Januarinfo. Nach der Auflösung in letzten Info las ich nun im Mumpitz 93 folgendes:

> **Aprilscherz**
>
> Bei dem in Heft 91 wiedergegebenen Bericht über Spectrum-Trojaner („Gelöchert") bin ich einem Aprilscherz aufgesessen. Hut ab vor dem Verfasser!

Yerzmyeys "Spreading-Service"



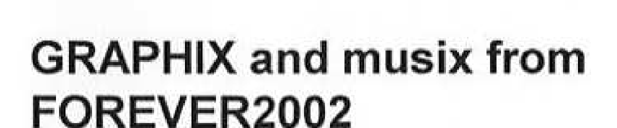


News: Demotopia goes TR-DOS

Sent: Mon, 1 Apr 2002 14:20:53 +0200

Hi all!
It's an interesting information from Gasman/ H-PRG to all Speccy lovers:
After 3 years online, Demotopia <http:// www.zxdemo.org/> has now entered a new stage... a new design which will include TR-DOS demos alongside the existing TAP downloads. With this new format, the site will do a better job of covering the full range of demoscene activity.
So far the TR-DOS additions are already available on "Virtual TR-DOS" <http:// zx.dotnet.lv/>, but I hope that future updates will introduce new demos not already found on the net, so that Demotopia will become the major demo archive for all sides of the scene.



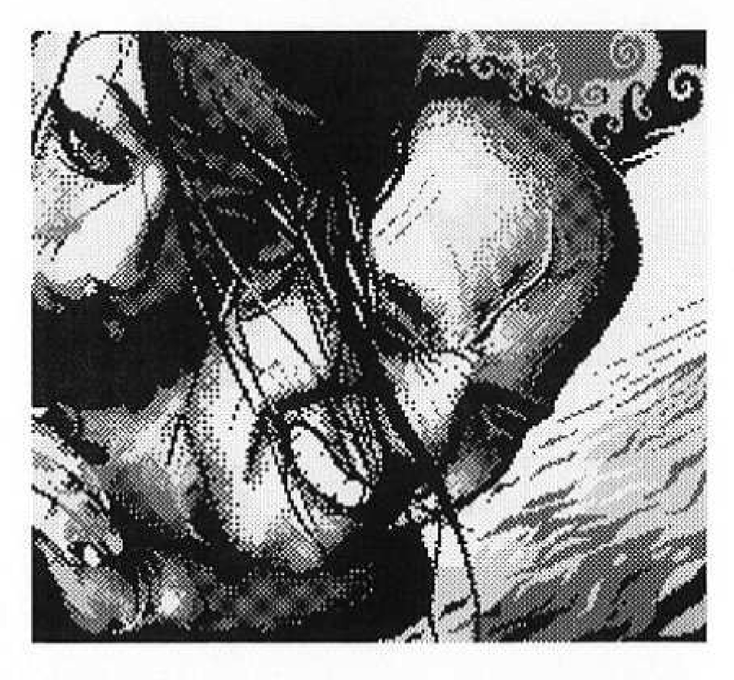


GRAPHIX and musix from FOREVER2002

Sent: : Fri, 5 Apr 2002 17:47:15

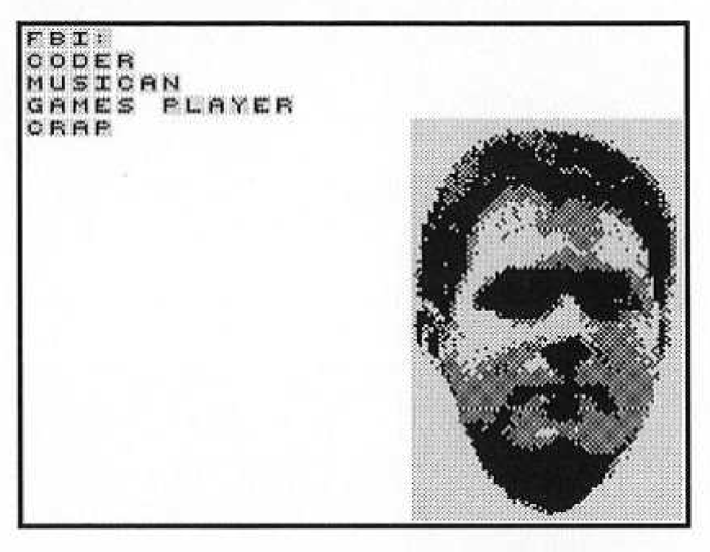
Hello!
Like in the subject. ;)
Byez, Y

(PS: Oky doky Yerz, here some printed pictures from Forever2002 party for all mag readers to show them, what ol' humble Speccy can do. Had a problem to include the great sounds here to paper. I guess it is better to send a ZIP-file to those, who are interested in more pics and tunes. Wo)



Demo from FOREVER 2002

Sent: Sat, 30 Mar 2002 16:23:58



Hi! This time take a look on "MISSING IN FUN" demo, a very nice old-school production, made by FBI and his friends. :)

ENJOY!!

Best wishes, Yerzmyey/H-PRG

(PS: Same her. If anybody wants Ebis demo, write to „womoteam@t-online.de". Wo)

TO all Spectrum Techno-Music-Makers - important

Sent: Thu, 2 May 2002 18:43:52

Hi, I organize a techno-music-project. I mean real music, namely techno-hardcore aka gabber. BUT - this project is related very strongly with Speccy, as I want to make this music using ZX.

The idea is to make on Spectrum main trax (bass, percussion and noices) and remix it on PC and synthesizers. I made one such a tune for now and it sounds quite interesting. All ready tunes I will upload on many MP3-sites (mp3.com etc...)

I will manage to make all musix by myself, but I am sure it would be much more interesting if it would be a regular techno-band.

So - if You have made some EXTREMLY hard and aggressive techno on Speccy, so send it to me (compiled, with basic loader/player). The tune cannot be looped. Add Your nickname and country. You will be listed as a band-member. But please - only TOTALLY hardcore trax.
If You don't know what is gabber exactly, take a look here:

http://click.mp3.com/c/f_990/u_artists.mp3s.com/artists/102/mortified_ragedj_gabber_go.html

If You are interested, just let me know on my e'mail address: ***hprg@poczta.fm***

Regards, Yerz

New zx-party in Poland

Sent: Wed, 22 May 2002 19:29:25

Hmmmmmm.
WILL (main organizer of ZX PARTY 2000) announced a new party with Speccy section. Take a look below:

City: Tuczno (near to Walcz city), next to Liptowskie lake. In a building of >>Osrodek "Lesny"<<. POLAND.

Time: 3-7 July 2002

Entrance fee only for Polish ppl: 40 pln (1$ = 4pln). Girls and foreigners can enter for free.

Info how to get there: ***silentriot@interia.pl***

Compos: Demo, intro 16, music, graphic.

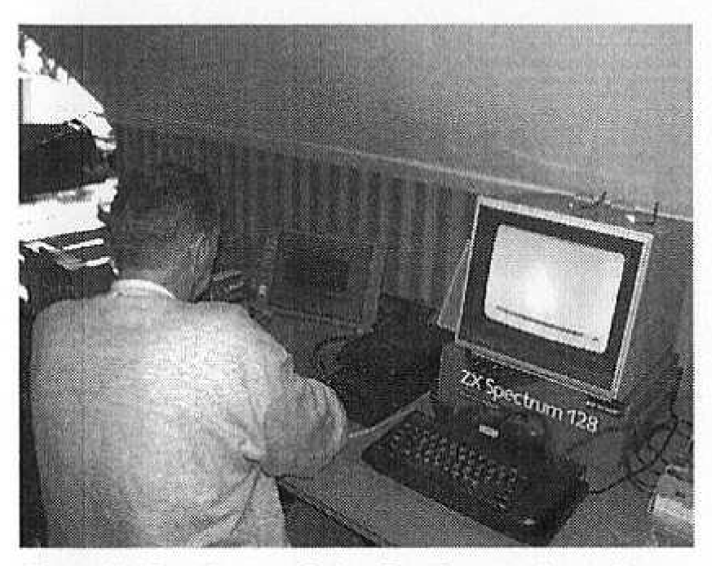
WWW of the party:
http://www.symphony.vdb.pl/

Compo machine - ZX128 and FDD3000 disc-drive.

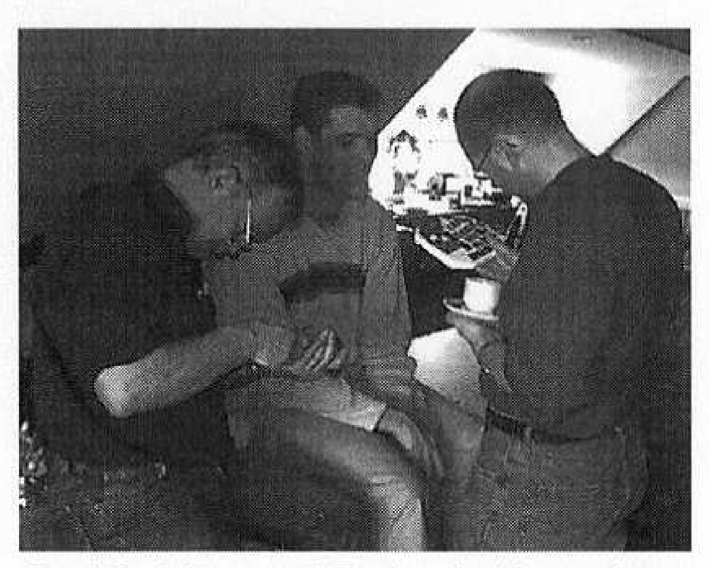
Send every Your works for the party to me. And BE FAST. If You have any questions about comming in person, write to "WILL" <doros@box43.pl>.

That´s all folks, bye Yerz

2. INTERNATIONALE SPECTRUM- UND SAM-TAGE IN URMOND AM 27. UND 28. APRIL 2002



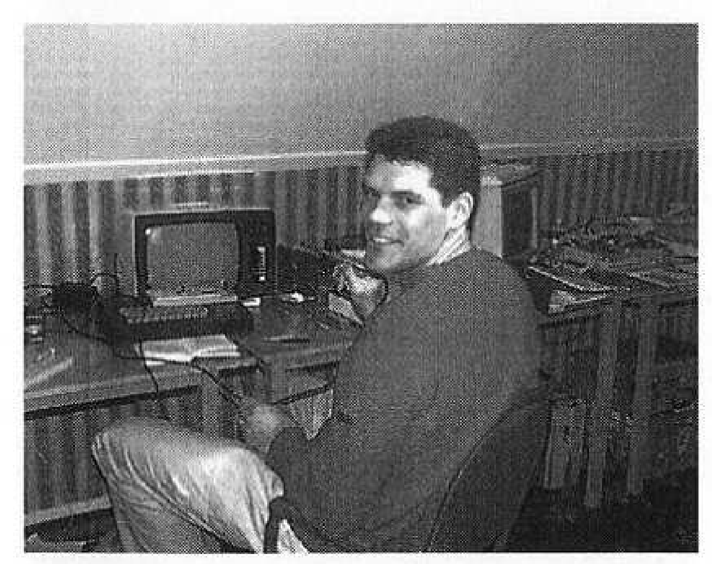
Ronald Raaijen auf der Suche nach weiteren Spectrum Programmen



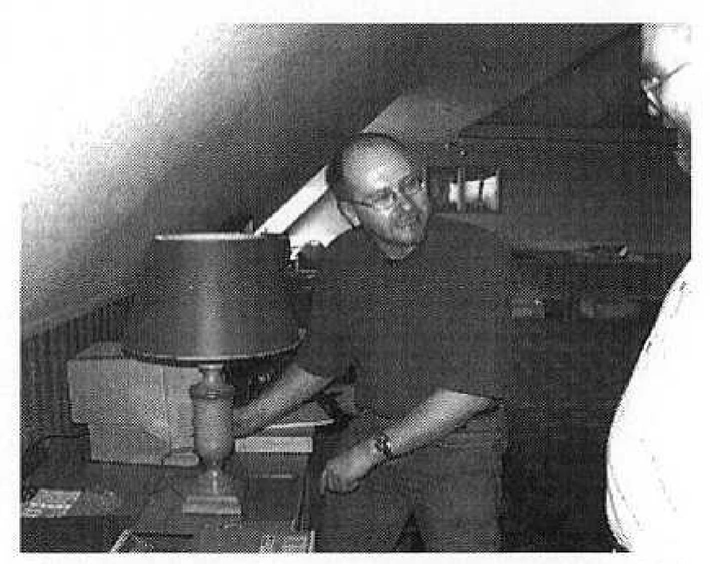
Roelof, Johan und Rudy - In Urmond war Holland dabei ;-)

Gegenüber dem Vorjahr waren es diesmal deutlich weniger Besucher, die den Weg nach Urmond auf sich genommen haben, um sich mit Gleichgesinnten zu treffen, zu klönen oder einfach nur über die „guten, alten Zeiten" zu schwelgen. Deshalb an dieser Stelle mein (und auch Johan Konings) herzliches Dankeschön an alle, die da waren.

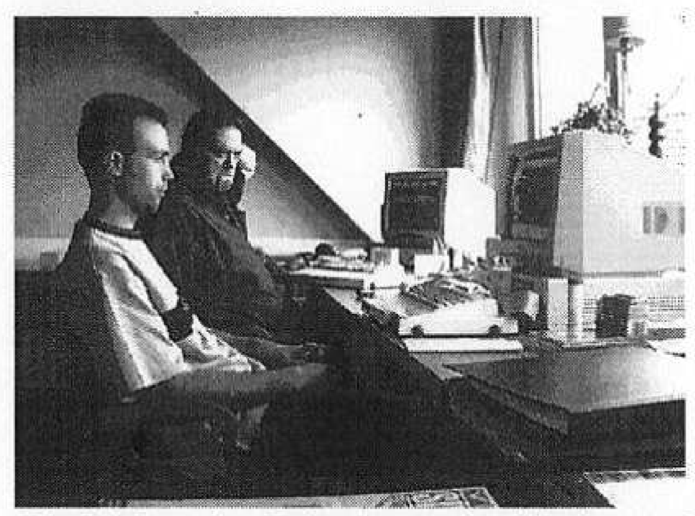
Wollen wir aber nicht unzufrieden sein. Immerhin waren ca.25 Besucher an beiden Tagen anwesend, das ist mehr als ein Drittel der Mitglieder. Trotzdem hätte ich mir für das Jubiläumsjahr des Spectrum etwas mehr gewünscht.
Schön das so ein Treffen doch immer wieder etwas neues bringt. Norbert Opitz hatte die

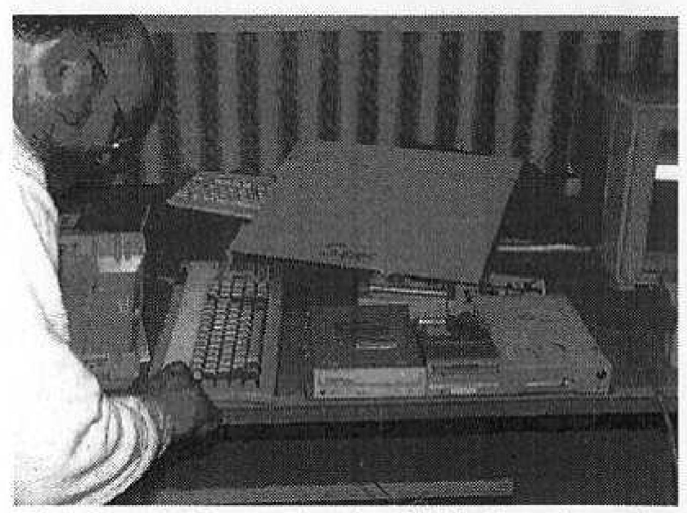


Johan Koelman mit Lötkolben und einem Lächeln im Gesicht...

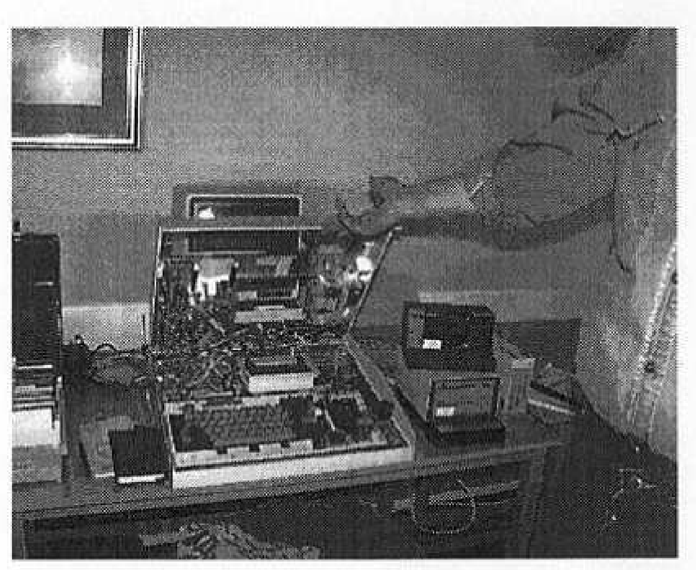


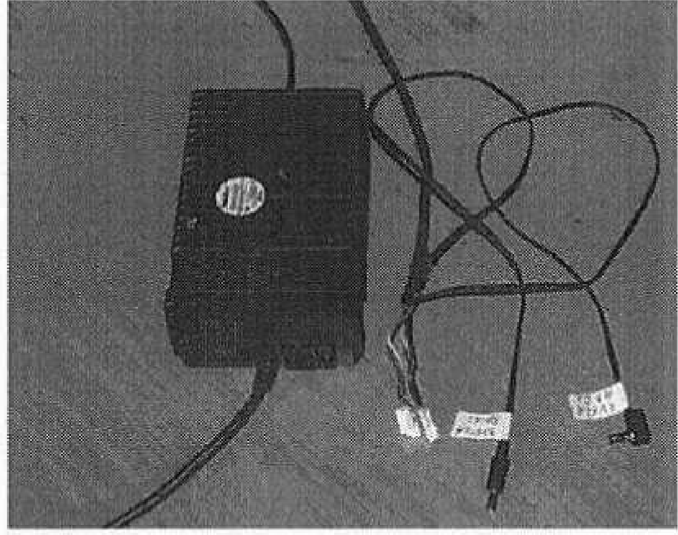
Rudy besorgte Dirk eine Lampe - da ging Dirk ein Licht auf

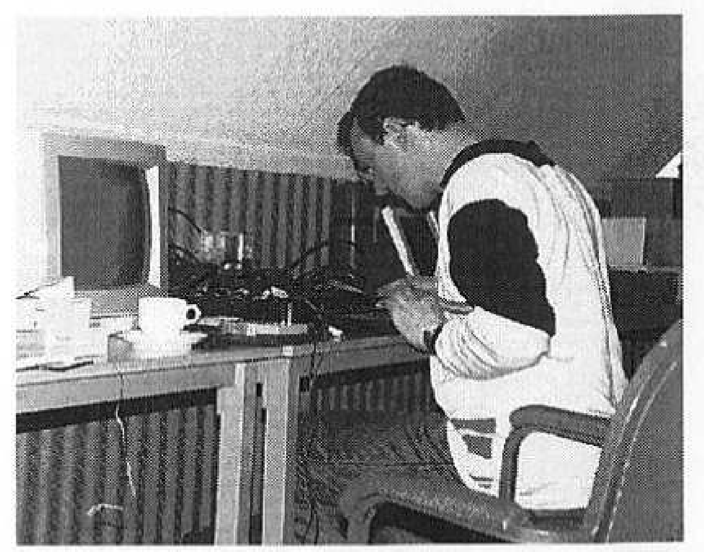

Sie fehlen nie: Robert van der Veeke und Martijn Groen, die SAM Spezialisten

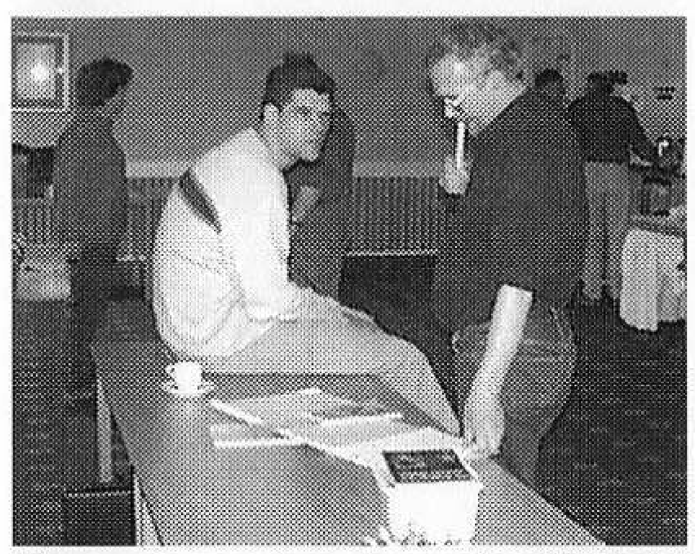

Dirk Berghöfer und sein SAM. So einen hat kein zweiter...


Liebes ZX-Team, schade, diesen Zeddy hat kaum einer von euch gesehen...


...eine sehr frühe Kompaktlösung wenn auch noch nicht mit Festplatte und LCD-Display


Norbert Opitz und sein Netzteil für MB02 und Spectrum in einem, lest dazu Seite 35


Wieder was fürs „Tagebuch eines Specci-Chaoten"? Diddi bei der „Arbeit"...



Mann, wie war denn das nochmal? Roelof beim Brainstorming...



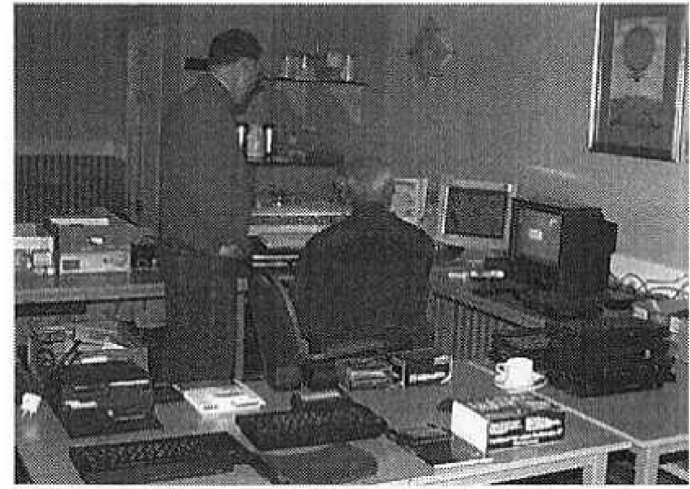
Seltener Anblick: Der Spectrum+3 von Peter Mai (reimt sich sogar)

Idee eines gemeinsamen Netzteils für seinen Spectrum und das MB02, ausserdem suchte er nach einer Lösung, das MB02 kostengünstig mit Speicherplatz zu erweitern. Verona hätte gesagt: „Hier werden Sie geholfen.“, dank Tipps und vieler Überlegungen seitens der Spectrum-Freunde gelangen Norbert die gewünschten Veränderungen. Seine Erfahrungen hat er nun für alle zum Nachlesen in zwei Artikeln in diesem Heft beschrieben.

Johan Koelmann hatte sein ZX<>PC Interface als ersten Versuchsaufbau mitgebracht und konnte so allen zeigen, wie wunderschön der PC als „Sklave“ für den Spectrum arbeitet. Eine Lösung, die sicher nicht nur mir zusagt, alleine schon aus Platzgründen bietet sie sich an. Das Interface soll nach Schätzung von Johan um die 15 Euro kosten. Johan zeigte sich auch bereit, auf Wunsch welche zu bauen, schließlich ist ja nicht jeder ein Hardwarebastler (das wird Heinz Schober sicher freuen, wenn er das liest). Auf diesem Treffen wurde auch gleich eine neue Idee geboren: Ein weiterer ZX81 Emulator. Auch dazu könnt ihr hier in diesem Heft mehr erfahren.

Peter Mai brachte seinen Spectrum +3 mit, einen Spectrum, den viele nur vom Hörensagen kennen. Er hatte das Gerät u.a. über e-bay ersteigert.

Ronald Raaijen brachte zum 20-jährigen Jubiläum seine inzwischen auf zwei CD´s an-

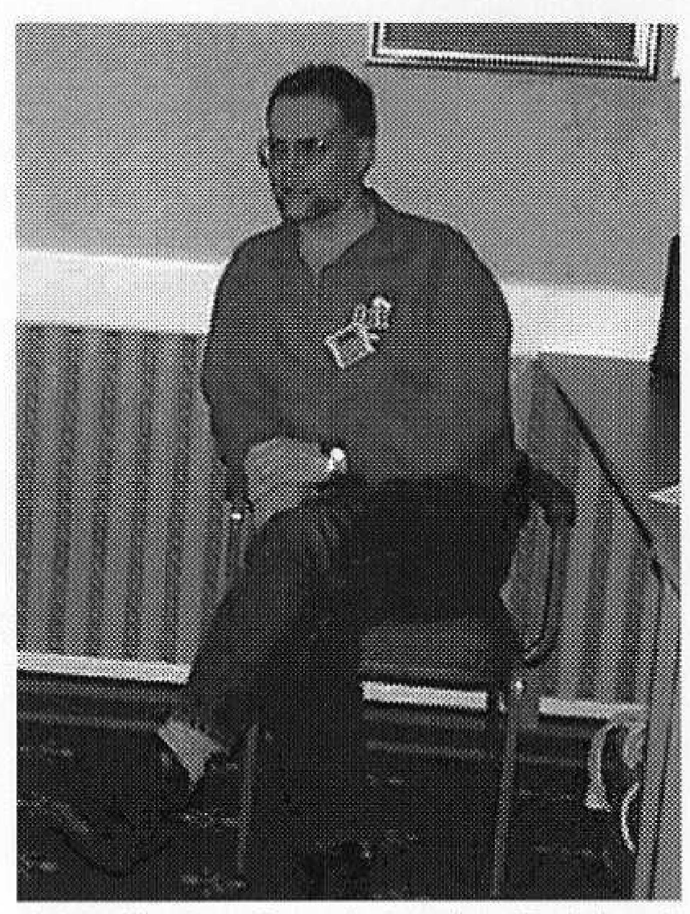


Wer suchet, der findet auch (meist) das was er braucht auf diesem Treffen

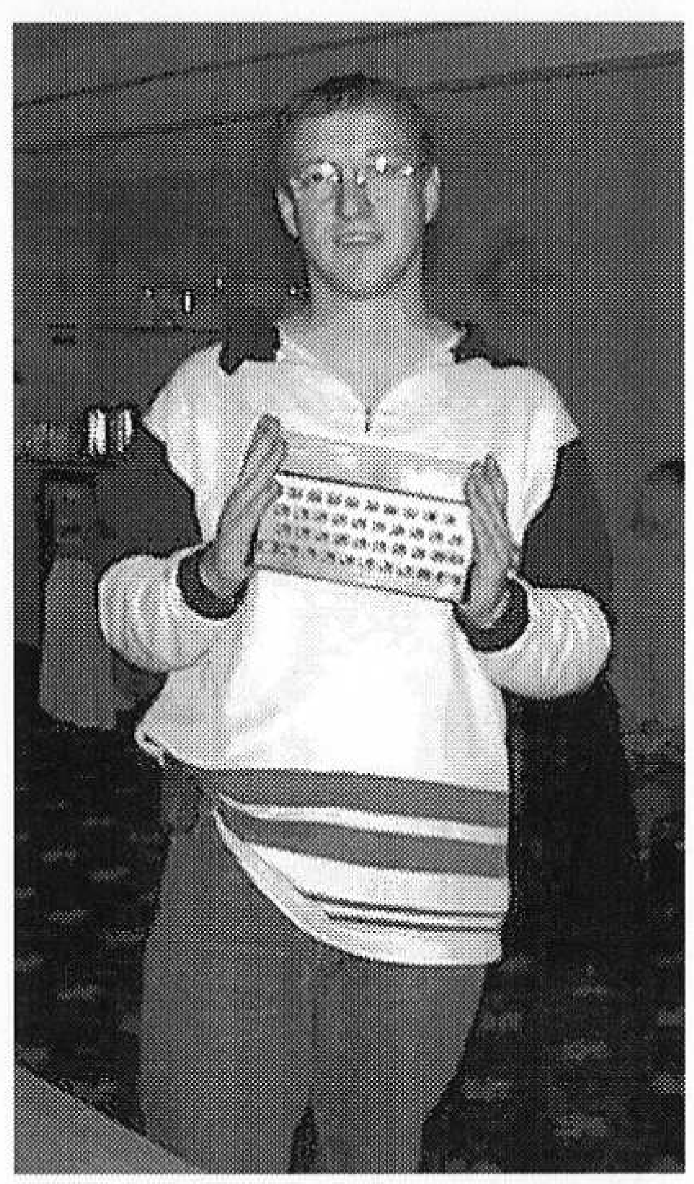


Philip Mulrane und Rudy Biesma, aber warum schaut Philip so seltsam auf den Speccy?

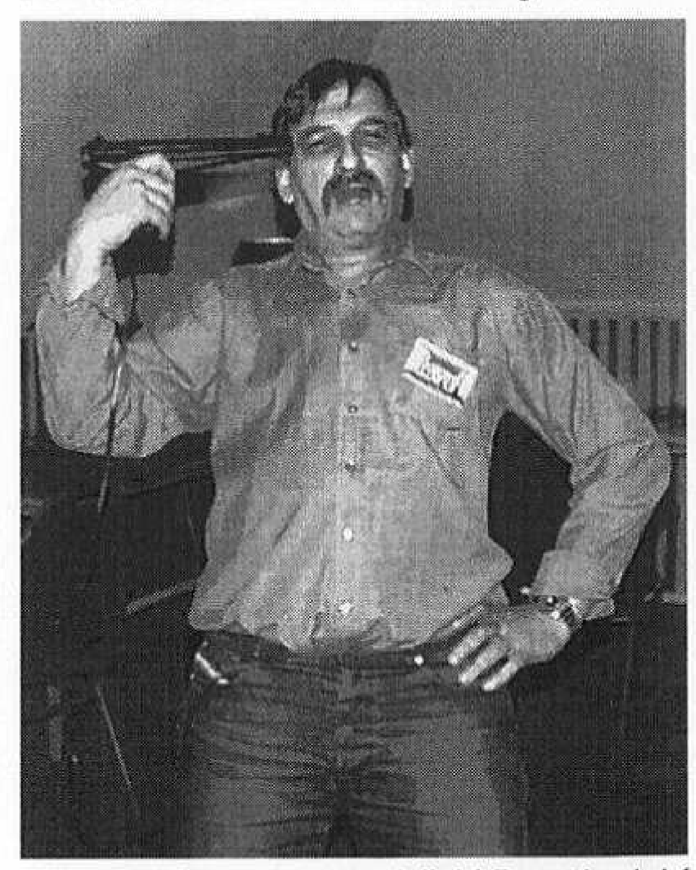
gestiegene „Sinclair Magic“ Sammlung mit - er bleibt auch weiterhin auf der Suche nach seltener Software. Bewundernswert.
Mir konnte beim Konvertieren von Opus-Disketten zum SAM geholfen werden. Das geht normalerweise mit Specmaker sehr gut, solange es sich um Basicfiles, Code oder Screen$ handelt. Mein Problem waren aber DATA a$() Files.
Martijn Groen arbeitet weiter an der DVD-Software für den SAM, die Sache scheint aber doch schwieriger zu sein.
Alle hatten wieder viel Spass und nach wie vor wechselten Soft- und Hardware den Besitzer. Fürs nächste Treffen wünsche ich mir dennoch ein paar Besucher mehr. *(Wo)*



Johan Koning, Organisator der „Party“, sah sich alles aus sicherer Entfernung an...



Diddi strahlt: Schaut mal her, ich hab wieder einen Specci zum „Verbraten“ gefunden... ;-)



The show is over - entsetzlich! Das überleb´ ich nicht...- sprach Wo und schoss

MCR-Generierung(9)

In diesem Teil wird die Assemblierung mit der Behandlung des 2. Passes fortgesetzt. Im Arbeitsbild vollzieht sich jetzt automatisch ein Dateienwechsel. Lesedatei wird jetzt die Datei ASMMITTE10 und Schreibdatei wird die Datei ASSELINK10. Bei den Tabellendateien werden die drei Dateien TDefSymb10, TExtSymb10 und TNazSymb10 gelesen, was durch die Angabe "EIN" verdeutlicht wird. Die Tabellendatei TUagSymb10 ist die einzige Schreibtabellendatei. Sie enthält die Symbole, deren Referenz nicht geschlossen werden konnte. Das heißt, daß es zu diesen Symbolen kein Merkmal als Bezugspunkt gibt. Das Protokoll der Befehle entspricht dem Inhalt der Datei ASSELINK10. Selbstverständlich sind bei den Anweisungen 00110, 00370 und 00380 die "*"-Kennzeichnungen noch nicht verschwunden. Das kann nur durch eine Korrektur geschehen. Aber es ist ein "*" in der Leerspalte vor der Befehlslängenangabe in der Anweisung 00140 dazugekommen. Das Symbol mit dem Namen "NEXBIT" im Operandenfeld des Befehles mit der Anweisungsnummer 00140 wurde in die Tabelle TUagSymb10 eingetragen, weil ihm kein Merkmal als Bezugspunkt zugeordnet werden konnte. Unter der Tabellenüberschrift "Unaufgelöste Referenzen" ist der entsprechende Ausdruck zu finden. Im 2. Pass können nur noch Fehler dieser Art auftreten. Diese Fehler sind nur durch eine entsprechende Korrektur zu beheben. Das ist die erste Aufgabe dieses Passes. Zum Ende des 2. Passes wird ein Fehler für den 2. Pass ausgewiesen. Damit erhöht sich die Gesamtfehleranzahl der Assemblierung auf sechs Fehler. Die zweite Aufgabe ist die Schließung der Referenzen, was man aus dem Befehlscode der Befehle Symbolen im Operandenfeld ersehen kann. Konnten die Referenzen noch nicht geschlossen werden, weil die Symbole als extern deklariert worden sind oder weil das Merkmal als Bezugspunkt in diesem Programmteil nicht definiert worden ist, dann zeigt sich das darin, daß der zweibytige Adressteil in der Kodierung dieses Befehles mit vier Nullen belegt ist. Das ist in den Befehlen mit den Anweisungsnummern 00030, 00120 und 00140 der Fall. In den Befehlen, in denen den Symbolen die entsprechenden Adressen, so wie sie durch den SPZZ vergeben worden sind, in hexadezimaler Form zugeordnet werden konnten, ist der Eintrag auch erfolgt. Das ist bei den Befehlen mit den Anweisungsnummrn 00160 für STRARR und 00210 für NEXVAR der Fall. So hat zum Beispiel der Befehl mit der Anweisungsnummer 00160 den Adreßeintrag 2B 00 erhalten, weil das Merkmal STRARR dezimal die Adresse 00043 durch den SPZZ erhielt. Das entspricht hexadezimal 2B00. Bei dem JR-Befehl in der Anweisungsnummer 00340 wird das berechnete relative Offset hexadezimal in das zweite Befehlsbyte eingetragen. Bei der Vorwärtsbezugnahme ist der Wert positiv und ist somit => 7F und bei Rückwärtsbezugnahme ist der Wert negativ und somit <= FF. Negative und positive Einbytewerte unterscheiden sich ja bekanntlich dadurch, daß bei positiven das höchste Bit den Wert 0 hat und bei negativen den Wert 1. Das Merkmal "PRINT" weist auf das erste Byte des Befehles mit der Anweisungsnummer 00190 und hat damit die Adresse 00038. Das zweite Byte des Befehles mit der Anweisungsnummer 00340, in das der Wert des Offsets eingetragen werden soll, hat die Adresse 00060. Die Adressendifferenz beträgt 22, das entspricht dem hexadezimalen Wert 16. Die hexadezimale Differenz von FF minus 16 ergibt das Offset E9. Das wird auch im Befehlsprotokoll als Eintrag zu dem Befehl "JR PRINT" ausgewiesen. Zum Abschluß wird das Arbeitsbild des 2. Passes aus dem Zwischenspeicher auf den Bildschirm transferiert, um die aufgelaufenen Satzzähler aktualisiert und ausgdruckt. Das Bild 1(9) zeigt das Arbeitsbild des 2. Passes. Damit ist die Assemblierung des ersten Beispieles abgeschlossen. Auf dem Protokoll ist ebenfalls die

```
Satzpro-     Z80URASSEM       2.Pass
tokoll                         1 Fehler
[0-3] : 3   #1.+2.Pass:  6 Fehler
PR u. LPR!    EINGABE    |  AUSGABE
  1. DATEI  Erg.1.pass   | Erg.2.Pass
Laufwerk:        1       |     2
Diskette:       185      |    186
Dat.-Name:  ASMMITTE10   | ASSELINK10
Satzanzahl    00042      |   00042
2.Tabellen  ASSEMB 2.Pass     Saetze
Laufwerk  | EIN: TDefSymb10    00007
    1     | EIN: TExtSymb10    00003
Diskette  | AUS: TUagSymb10    00001
   186    | EIN: TNazSymb10    00006
Datum: 07.05.2001 Uhrzeit: 09.39
FEHLER:
```

Bild 1(9)

Programmlänge in Bytes angegeben. Sie ist eine wichtige Angabe bei den Parametern der zu verbindenden Programme.

Auf Grund der aufgetretenen Fehler ist es notwendig, einen Korrekturlauf durchzuführen. Das Ziel sollte dann aber auch sein, alle aufgetretenen Fehler zu beheben, damit die Gesamtfehleranzahl der Assemblierung 0 ist. Die Korrekturbefehlsfolge könnte wie folgt aussehen:

`110 -> BIT 7,A`
Doppeltes Merkmal aufheben!

`350 -> EXTRN VARANF,BIT5`
Merkmal BEGINN darf nicht EXTRN deklariert sein!

`360 -> ENTRY BRACKE`
NEXBIT tritt in diesem Programmteil nicht als Merkmal auf!

`370 -> ENTRY POINTE`
Merkmal BRACKE darf nicht doppelt als ENTRY deklariert sein!

`380 -> LOE`
Symbol VARANF darf nicht doppelt als EXTRN deklariert sein!

`390 -> EXTRN NEXBIT`
NEXBIT tritt aber als Symbol auf und muß als EXTRN deklariert werden !

Es wäre auch möglich, an Stelle des LOE-Befehles auf 380 gleich den 380-er Inhalt mit EXTRN NEXBIT zu überschreiben. Auch hätte man 360 und 370 zu 360 ENTRY BRACKE, POINTE zusammenziehen können und 370 dafür mit LOE belegen können. Das zeigt, daß es gerade im Bereich dieser beiden Anweisungstypen durchaus unterschiedliche Korrekturmöglichkeiten gibt. Diese Befehlsfolge könnte mit Z80UKEDITI in die Datei Z80UKTEST2 eingegeben werden. Dann transferiert Z80UKTRANS diese Befehlsfolge in die Datei EDITKORR02. Im Korrekturlauf wird danach die alte Ur-Datei EDITASSE10 mit der Korrekturdatei EDITKORR02 zur neuen Ur-Datei EDITASSE12 verarbeitet. Der letzte Gang ist die Assemblierung der neuen Ur-Datei EDITASSE12. Bei dieser Assemblierung könnte bei der Schreibversionsnummernabfrage auch wieder die 10 eingegeben werden. Dann würden alle 10-er Dateien und Tabellendateien nach Erlaubnis überschrieben werden und man bleibt auf dem 10-er Weg. Aber ebenso ist auch ein neuer Weg mit der 10 denkbar. Die durchführung der Assemblierung liefert das gewünschte Ergebnis. Es werden 0 Fehler gemeldet.

Zunächst noch eine Anmerkung zu den Protokollausdrucken. In den einzelnen Artikeln dieser Serie werden nur die Protokolle der Arbeitsbilder dargestellt, nicht die Befehls- und Fehlerprotokolle. Hier setze ich voraus, daß der Anwender, genau wie ich es auch gemacht habe, mit den vorgegebenen Befehlsfolgen die Programmläufe selbst durchführt. Dann erhält er die gewünschten Befehls- und Fehlerprotokolle. Und noch ein Hinweis: Wenn z.B. der 1. Pass 4 Fehler gemeldet hat und man unterbricht den weiteren Programmlauf, um dann später sofort mit dem 2. Pass zu beginnen, kann selbstverständlich der 2. Pass von den 4 Fehlern aus dem 1. Pass nichts wissen. Bringt jetzt der 2. Pass 0 Fehler, dann wird natürlich die Gesamtassemblierung mit 0 Fehlern ausgewiesen. Es sind also in einem solchem Falle die Fehler des 1. Passes von Hand zu denen des 2. Passes dazuzurechnen, um die Gesamtfehleranzahl zu erhalten.

Wenn der 1. Pass beendet ist und aus zeitlichen Gründen oder wegen eines Diskettenfehlers oder Rechnerabsturzes der 2. Pass nicht beendet werden konnte, ist es möglich,

den Assemblerlauf gleich mit dem 2. Pass zu beginnen. Dazu ist die Frage, ob mit dem 1. Oder 2. Pass begonnen werden soll, mit 2 zu beantworten. Aus dem Protokolldruck des Arbeitsbildes des 1. Passes ist bekannt, auf welchen Disketten sich die Tabellendateien und die Datei ASMITTExx befinden, so daß in das Arbeitsbild des 2. Passes die korrekten Angaben eingegeben werden können. Selbsverständlich ist auch darauf zu achten, daß die Disketten in die angegebenen Laufwerke eingelegt werden. Sonst kann die Koordinierungsfehlermeldung bzw. die Meldung "Datei nicht gefunden!" auf-treten. Der nun folgende Ablauf ist oben schon beschrieben worden. Die Datei ASMITTExx fungiert nur als Zwischendatei.

```
Satzpro-      Z80URASSEM      1.Pass
tokoll                        4  Fehler
[0-3] : 3        PRL: 45  Bytes
PR U. LPR!     EINGABE       AUSGABE
  1. DATEI   Urs.-Datei  | Erg.1.Pass
Laufwerk:         2      |     1
Diskette:        186     |    185
Dat.-Name:  EDITASSE11   | ASMMITTE11
Satzanzahl      00026    |   00026
2.AUSGABE-Tabellen ASSEMB  Saetze
                  | TDefSymb11    00007
Laufwerk:    1    | TExtSymb11    00002
                  | TEntSymb11    00002
Diskette:  185    | TNazSymb11    00007
Datum: 13.05.2001 Uhrzeit:  17.39
FEHLER:
```

Bild 2(9)

Nun zum zweiten Beispiel. Alles verläuft analog zum ersten Beispiel. Die Bilder 2(9) und 3(9) zeigen die Ausdrucke der Arbeitsbilder des 1. und 2. Passes. Auch hier weist das Gesamtdruckprotokoll Fehlermeldungen auf, die analysiert und korrigiert werden müssen. Der Protokolldruck weist aus, daß der 1. Pass 4 Fehler gemeldet hat. Der 2. Pass ist mit 0 Fehlern beendet worden. Also 4 Fehler in der Gesamtassemblierung. Diese Fehler treten nicht direkt im ersten Protokollteil des 1. Passes auf, sondern erst bei der Prüfung der Tabellenbeziehungen. Die erste Prüfung "Unerlaubte Def. Ext. Symbol!" ist leer und damit fehlerlos. Die zweite Prüfung "Verboten ext. = int. Symbol!" enthält zwei Fehlermeldungen. Beide Symbole sind

```
Satzpro-      Z80URASSEM      2.Pass
tokoll                        0  Fehler
[0-3] : 3  #1.+2.Pass:    4  Fehler
PR U. LPR!     EINGABE    |  AUSGABE
  1. DATEI   Erg.1.pass   | Erg.2.Pass
Laufwerk:         1       |     2
Diskette:        185      |    186
Dat.-Name:  ASMMITTE11    | ASSELINK11
Satzanzahl      00026     |   00026
2.Tabellen  ASSEMB 2.Pass     Saetze
Laufwerk | EIN: TDefSymb11    00007
    1    | EIN: TExtSymb11    00002
Diskette | AUS: TUagSymb11    00000
   185   | EIN: TNazSymb11    00007
Datum: 13.05.2001 Uhrzeit:  17.39
FEHLER:
```

Bild 3(9)

sowohl als extern als auch als intern (ENTRY) deklariert worden. Die dritte Prüfung ""Int. Symbol nicht definiert!" enthält deshalb dieselben Namen als Fehlermeldungen, weil sie als intern deklarierte Namen nicht auch gleichzeitig als Merkmal in diesem Programmteil definiert worden sind. Daß beide Namen in diesem Programmteil als Symbole in Operandenfeldern vorkommen, ist richtig und daher ist die ENTRY-Anweisung für sie falsch. Mit dem Korrekturbefehl 240 -> LOE sind alle 4 Fehler mit einem Schlag behoben. Der Korrekturbefehl wird mittels Z80UKEDITI in die Datei Z80UKTEST3 eingegeben und anschließend mittels Z80UKTRANS in die Specci-Datei EDITKORR03 transferiert. Mit dem Korrekturprogramm Z80URKORR wird aus der alten Ur-Datei EDITASSE11 durch die Korrekturdatei EDITKORR03 die neue Ur-Datei EDITASSE13 erzeugt. Wenn der Anwender diese Programmläufe ausführt, werden 0 Fehler in der Assemblierung am Ende zu Buche stehen. Die Ergebnisdatei ist in

```
Satzpro-      Z80URASSEM      1.Pass
tokoll                        0  Fehler
[0-3] : 3        PRL: 45  Bytes
PR U. LPR!     EINGABE   |   AUSGABE
  1. DATEI   Urs.-Datei  | Erg.1.Pass
Laufwerk:         1      |     2
Diskette:        185     |    186
Dat.-Name:  EDITASSE13   | ASMMITTE13
Satzanzahl      00024    |   00024
2.AUSGABE-Tabellen ASSEMB  Saetze
                  | TDefSymb13    00007
Laufwerk:    2    | TExtSymb13    00002
                  | TEntSymb13    00000
Diskette:  186    | TNazSymb13    00007
Datum: 15.05.2001 Uhrzeit:  13.55
FEHLER:
```

Bild 4(9)

```
Satzpro-      Z80URASSEM      2.Pass
tokoll                      0  Fehler
[0-3] : 3   #1.+2.Pass:  0 Fehler
PR u. LPR!    EINGABE    |  AUSGABE
  1. DATEI  Erg.1.pass   | Erg.2.Pass
Laufwerk:        2       |     1
Diskette:       186      |    185
Dat.-Name:  ASMMITTE13   | ASSELINK13
Satzanzahl     00024     |   00024
2.Tabellen  ASSEMB 2.Pass     Saetze
Laufwerk | EIN:  TDefSymb13    00007
   2     | EIN:  TExtSymb13    00002
Diskette | AUS:  TUagSymb13    00000
  186    | EIN:  TNazSymb13    00007
Datum:  15.05.2001 Uhrzeit:  13.55
```

Bild 5(9)

meinem Falle die Datei ASSELINK13. Für die Assemblierung des ersten Programmteiles ist es die Datei ASSELINK11. Die Bilder 4(9) und 5(9) zeigen die Ausdrucke der Arbeitsbilder des 1. bzw. 2. Passes. Auch hier ist im Protokoll die Programmteillänge von 45 Bytes vermerkt. Das Wichtigste bei der Assemblierung ist zu wissen, wie die gemeldeten Fehler behoben werden können. Damit wären die wesentlichen Dinge zum Assemblierungsprogramm Z80URASSEM gesagt worden.

```
Satzpro-      Z80OBLISTE   v-A. 00000
tokoll                     b-A. 00555
[0-3] : 2                  Dateiart o
Nur LPR!      EINGABE    |  AUSGABE
  1. DATEI  Druckdatei   |
Laufwerk:        2       |
Diskette:       185      |
Dat.-Name:  ASSELINK13   |
Satzanzahl     00024     |
  2. DATEI               |
Laufwerk:                |
Diskette:                |
Dat.-Name:               |
Satzanzahl               |
Datum:  16.05.2001 Uhrzeit:  10.33
```

Bild 6(9)

Als nächtes wollen wir uns dem Programm Z80OBLISTE zuwenden, das den Nebenprogrammen zuzuordnen ist. Es arbeitet in gleicher Weise, wie das Programm Z80URDRUCK und ist mit seiner Bedienung identisch. Lesedateien sind alle ASSELINKxx-Dateien. Im Ergebnis entsteht auch hier ein Listenausdruck. Das Arbeitsbild dazu zeigt das Bild 6(9).

(Fortsetzung folgt!)

Erwin Müller
Strehlener Straße 6B, 01069 Dresden

Ein neues Magazin: „Retro Review“



Die ersten beiden Ausgaben des jeweils 56-seitigen Magazines

Anfang Mai erhielt ich ein Mail aus Portugal mit dem Hinweis auf diese Webseite:

http://www.retroreview.com

Nach einem Ausflug dorthin war meine Neugier sehr groß geworden und ich mailte den Herausgeber, Jorge Canelhas an. Das Ergebnis: Jorge ist nun Mitglied im SPC und wir tauschen unsere Magazine aus.

Inzwischen liegen mir die ersten beiden Ausgaben vor. Und gleich vorweg: Die Magazine sind Spitze und sehr professionell gemacht. Hauptthemen sind ältere Computersysteme, aber auch Spielreviews und Interviews. Ausserdem gibt es eine Auktion, über die man alte Systeme erhalten kann. Der Spectrum kommt in beiden Heften sehr häufig vor, ebenso der ZX81 und Timex 2048. Ausserhalb der Sinclairs findet man Verschiedenes zum C64, Atari und Amiga.

Das Heft erscheint alle 2 Monate zu einem Einzelpreis von 4 Euro (plus 2 Euro Versand).

Interessierte wenden sich bitte an:

Jorge Canelhas, Apartado 3115
Miguel Bombarda, P-2745 Queluz, Portugal
Mail: jcanelhas@retroreview.com
Web: http://www.retroreview.com



So, 12.Okt. 86

Haben gestern Onkel Hubertus besucht. Er ist der Bruder meiner Mutter, ich mag ihn eigentlich ganz gerne, nur wir sehen uns selten, weil er weit weg wohnt. Ab und zu gibt er mir am Telefon ein paar Tips, er ist nämlich Ingenieur und weiß viele Sachen über Physik.

Nun lag er im Krankenhaus, also sind wir die 140 KM zu ihm gefahren um ihn zu besuchen. Es ging ihm ganz gut, ich hab gesehen daß er heftig mit den Schwestern dort geflirtet hat, naja, er ist ja erst 38 Jahre alt .....

Jedenfalls haben wir uns natürlich darüber unterhalten warum er im Krankenhaus ist. Er sagte „Wegen dem blöden Murphy-Gesetz". Ich wußte schon einiges darüber, aber mein Papa nicht. So erzählte mein Onkel, daß Murphys Gesetz sagt, *„daß alles schiefgeht, was schiefgehen kann"*. Und andere Leute haben Ableitungen davon erstellt. Wir lachten, aber Onkel Hubertus sagte, daß wir es ruhig glauben könnten, es würde stimmen. Er sagte daß er ein Buch voll von solchen Gesetzen habe, und die würden stimmen. Oliviers Ableitung sagt zum Beispiel *„Erfahrung ist eine äußerst nützliche Sache, leider hat man sie immer erst kurz* ***nachdem*** *man sie braucht!"*

Er hatte die Holzkonstruktion vom Dachstuhl reparieren wollen und hatte sich ein Brett über die Sparren drübergelegt, um an die Stellen zu gelangen die er ausbessern wollte. Das Brett war naürlich länger als die Sparren, so stand es über den Rand hinaus. Er grinste schief und sagte, die Erfahrung die er jetzt hätte, besteht darin, daß er nicht über das Ende eine Bohle treten darf, das über einen Balken übersteht, da es sonst hochschnellt - diese Erfahrung hätte er kurz davor noch nicht gehabt. Erinnerte mich ein bißchen an die Bierzelte, wenn da jemand ganz am Rand einer Bank sitzt und der letzte steht auf, fliegt der andere auf den Boden! Na jedenfalls muß mein Onkel zwei Tage dableiben, sie wollen röntgen und sichergehen daß er sich bei dem Sturz keine inneren Verletzungen geholt hat.

Mo, 13.Okt. 86

Hab über das nachgedacht, was Onkel Hubertus erzählt hat. Es müßte doch möglich sein, die Wahrscheinlichkeit von Murphys Gesetz zu berechnen. Wenn ich das schaffe, dann ist Murphys Gesetz durchschaubar, man weiß schon, wann es zuschlägt. Gefällt mir, der Gedanke. Wolle gerade schon mal die erste Formel zu Papier bringen, aber meine Mutter rief nach mir, weil ich ihr etwas helfen soll. Verschiebe es auf später.

Do, 16. Okt.86

Also im Moment komme ich nicht dazu, mir eine erste Formel aufzuschreiben. Immer wenn ich grade soweit bin, kommt irgendwas dazwischen. Einmal war die Kulimine in dem Moment leergegangen, als ich die Formel aufschreiben wollte. Naja es läuft mir ja nicht weg.

Sa. 18. Okt. 86

Jello ist mal wieder hier. Mein Freund Jello und ich spielen immer unheimlich gerne Maniac Maziacs, wo wir abwechselnd versuchen, das Labyrinth zu erforschen, Gold einzusammeln und vor diesen ekligen Spinnenviechern abzuhauen. Er erzählte mir daß er auch mal versucht habe, Murphys Gesetz zu erforschen, aber er habe aufgegeben, weil er seine Gesundheit nicht aufs Spiel setzen

wolle. Das verstand ich nicht, aber alle Versuche, von ihm mehr zu erfahren, waren zwecklos.

Di, 21. Okt. 86

Hab heute ein bißchen Basic programmiert, während meine Mutter dachte, ich mache meine Kunstunterricht-Hausaufgaben. Wenn die wüßte! Ich geh immer zu Sammy, meinem kleinen Bruder, und guck mir seine Zeichnungen an. Der malt immer so ein Zeug, da kannst du ALLES draus erkennen, wenn du nur willst. Ich frag ihn dann ob er mir eins gibt, meistens will er von mir dafür an den Spectrum dran gelassen werden. Fairer Tausch. Jedenfalls nehm ich dann diese Zeichnung in die Schule, geb sie für meine aus, und wenn der Lehrer sagt, daß wir doch Häuser malen sollten, sage ich nur „sehen Sie bitte diese abstrakten Linien, sie geben die Tücken des Verfalls wieder, dem unsere Häuser ausgesetzt sind....." ich hab so Sätze mal in einem Kunstbuch gelesen, mein Lehrer ist immer fasziniert, und gibt mir meistens eine 2, hihi!
Als ich also Basic programmierte, (ich bin grad an einem Programm, um den Calculator im Spectrum wie einen Taschenrechner zu bedienen), kam mir eine Idee. Welche Stackreihenfolge müßte ich eigentlich abspeichern, um die Varianz der Risiken aufzurechnen, die....... der Bildschirm von meinem Spectrum wurde schwarz, und dann kam die Einschaltmeldung - alles weg!!!! Und ich hattte nicht gespeichert! Das kann doch nicht sein!! Sollte...... nein das kann nicht sein! Murphys Gesetz schützt sich selbst? Immer wenn jemand es zu knacken versucht, gibt es Pech? So nicht! Nicht mit mir! Murphy, du wirst durchschaut!!

Mi, 22. Okt. 86

Habe mit Jello telefoniert. Er sagt daß er genau zu diesem Schluß gekommen ist. Er erzählte mir, wie er auch mal versucht hatte, dieses Gesetz zu berechnen, und sein damaliger Computer, ein Commodore 64, ging in Rauch auf, deshalb hatte er ja danach einen Spectrum, der ist zuverlässiger. Hatte es aber nicht wieder riskiert. Pfff, Feigling.
Ich setzte mich an meinen Spectrum, schaltete ihn ein, und schrieb die erste Basic Zeile

10 REM Murphologie Berechnung

Dann startete ich das Tape, um diese eine Zeile zu sichern, wollen doch mal sehen, ob ich diesem Gesetz nicht zeigen kann, wer hier Herr ist. Das Band lief, ich gab ein

SAVE „MURPH1" LINE 10

drückte ENTER, ... in dem Moment riß das Band im Recorder, es gab Bandsalat, ich war für den Rest des Nachmittages mit dem auseinandernehmen und reinigen des Recorders beschäftigt.
und dauernd grinste auf dem Bildschirm

0 OK; 0:1

könnte kochen vor Wut !!!!!!

Sa, 25.10.86

Jello war zu Besuch, zusammen mit Toni, dessen Vater eine Pizzeria hier im Ort betreibt. Wir kamen auf Jellos Pech mit dem Commodore 64 zu sprechen. Toni hat auch einen C64, und wir ärgerten uns gegenseitig mit den Fähigkeiten unserer Computer. Wozu braucht mein ZX Spectrum denn Sprites? Der zeichnet wenigstens locker einen Kreis mit Circle, mach das mal mit deinem Brotkasten, Toni!
Jello bremste uns etwas und meinte, eines Tages würden sich sogar Commodore- und Sinclair- und PC-User vertragen und die Rechner zusammenschalten. Ich lachte ihn aus, das passiert doch nie!

Di, 28. Okt. 86

Saß wieder vor dem Spectrum. Ich wollte meinem Vater zeigen, wie unglaublich Mur-

phys Gesetz ist. Ich machte genau dasselbe wie gestern, REM Zeile schreiben, SAVE ,.... diesmal klappte es! Ich dachte an eine Bemerkung von Jello, daß nämlich die Ableitung von Willboughby's Gestz sagt: *„Wenn man jemandem zeigen will, daß eine Maschine nicht funktioniert, dann tut sie's !"*
Ich grinste innerlich und dachte, daß ich ja dann auch gleich weitermachen kann, vielleicht traut sich diese Pechsträhne nicht, zuzuschlagen, solange mein Papa hier ist...
Papa sagte, daß er mir kurz zuguckt, wie ich programmiere. Ich stellte den Spectrum extra auf den großen Tisch in meinem Zimmer, damit wir mehr Platz hatten, nur daß das Netzkabel von der Fenstersteckdose her ziemlich gespannt war, weil es zu kurz für die Entfernung war. Während ich die vierte Formel eingab (und dabei nicht SAVE machte, Papa war ja da.....) kam meine Mutter ins Zimmer um Staub zu wischen. Ich achtete nicht weiter auf sie. Aber in dem Moment, als ich zu meinem Vater sagte „Guck mal jetzt geht alles glatt......" da stolperte meine Mutter über das Netzkabel, das gespannt zwischen der Steckdose am Fenster und meinem Tisch verlief !!!

Mi. 29. Okt. 86

Noch ein Versuch! Diesmal wartete ich, bis alle aus dem Haus waren. Nahm zwei Kassettenrecorder, zwei verschiedene Kassetten, holte dafür extra Vaters Dreifachsteckdose vom Fernseher weg, schloß mich in mein Zimmer ein nachdem ich durch die Wohnung gegangen war und alle Wasserhähne, Netzkabel, Türen und Fenster kontrolliert hatte. Ich war wütend! Es wäre doch gelacht wenn ich diesem dämlichen Gesetz nicht eine lange Nase zeigen könnte. Wild begann ich drauflosschreiben in Basic, schimpfte und drohte unentwegt, machte alle 4 Minuten SAVE, abwechselnd auf beide Recorder, und probierte zwischendurch mit RUN, ob auch keine Fehler im Programm waren. Die Formel entstand, ich kam wirklich gut voran. Nach 4 Stunden hattte ich etwa 3000 Basiczeilen, die die Wahrscheinlichkeit von Murphys Gesetz immer genauer berechnen sollten. Endlich war ich soweit, die fertige Version abzuspeichern. Ich sicherte sie auf beide Recorder. Uff das war geschafft. Stolz und unglaublich glücklich ging ich in die Küche um mir etwas zu trinken zu holen. Ich kam zurück, und sah daß der Spectrum nicht abgestürzt war. Na also, geht doch!
Ich steckte das Kabel von SAVE auf LOAD um, um einen VERIFY zu machen. Auf dem ersten Band - NICHTS. Kein Laut kein Ton, kein BEEP, einfach nur leises Rauschen! PANIK!
Kabel in den zweiten Recorder, NICHTS. Kein Laut, kein Ton etc.......
Jetzt nur nichts falsch machen! Was war passiert? Vorsichtig untersuchte ich das Kabel, denn ich wußte, die einzige echte vorhandene Version meine Programmes war eben auf keiner Kassette, sondern nur im RAM meines Spectrum! Das Kabel hatte am Stecker einen Wackelkontakt. Ich probierte nochmal ein SAVE, hielt dabei das Kabel ein wenig hoch, leider ohne Erfolg. Auf meinem Spectrum Bildschirm stand jetzt als letzte Zeile

SAVE „MURPH76" LINE 10
0 OK 0:1

Aber auf dem Kassettenrecorder war wieder nichts drauf! Auf Zehenspitzen schlich ich mit dem Kabel in den Keller, um es zu löten. Während ich da unten war, kamen meine Eltern heim. Zehn Minuten später kam ich mit dem reparierten Kabel in mein Zimmer ——— und mir stockte der Atem! Mein Spectrum war dunkel, alles war ausgeschaltet. Ich lief zu meinem Vater, der mich stocksauer anschaute! Er sagte, daß ich ihm eine Videoaufnahme ruiniert hätte, an der Dreifachsteckdose hatte auch der Videorecorder gehangen, und der hatte nicht aufgenommen! Er habe sich die Dreifachsteckdose wiedergeholt, ich hätte ja alles gespeichert, **wie er an der letzten Spectrum-**

zeile gesehen hätte, und ich sollte mir ein Beispiel nehmen, daß er wenigstens vorher guckt, bevor er einfach eine Dreifachsteckdose entfernt!

Do, 30. Okt 86

Ich gebe auf! Murphys Gesetz stimmt bis ins letzte Detail!

Anmerkung des Autoren: diese Tagebuchversion ist mit 40 SAVE-Vorgängen erstellt worden ......

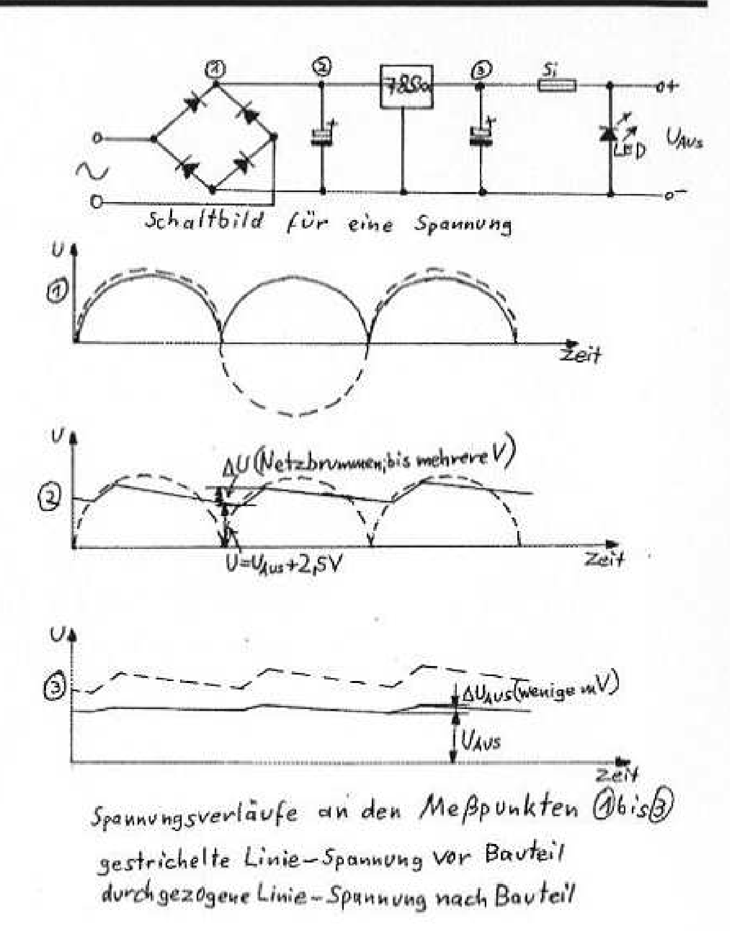
Netzteil für MB 02 und Spectrum

Auf dem Treffen in Aidlingen sind mir zwei Netzteile des Spectrum +3 für das MB 02 abgeraucht, der Grund ist mir bis heute nicht klar. So habe ich nach einem Einblick in die Netzteile zur Radikalkur entschlossen und habe die Elektronik entfernt. Der vorhandene „H"-förmige Kühlkörper und der vorhandene Raum brachte mich auf meine alte Idee, MB 02 und Spectrum aus einem Netzteil zu betreiben.

Von der alten Leiterplatte ist nur ein Streifen mit den Trafoanschlüssen übrig geblieben und der „H"-förmige Kühlkörper ist in seine „Striche" geteilt. Als neue Leiterplatte habe ich eine Lochrasterplatte mit Lötaugen genommen.

Das Netzteil erzeugt nun drei Spannungen, 5V / 2A und 12V / 0,7A für MB 02 sowie 7,5V / 2A für den Spectrum.

Die 7,5V für den Spectrum sind ausreichend, da die Orginal-Netzteile ohne Spannungsstabilisierung auf Funktionssicherheit und erschwinglichen Preis ausgelegt waren. Durch die niedrigere zugeführte Spannung entsteht im Spectrum nicht mehr soviel Wärme.



Spannungsverläufe an den Meßpunkten 1 bis 3
gestrichelte Linie – Spannung vor Bauteil
durchgezogene Linie – Spannung nach Bauteil

Die jeweiligen Spannungswege sind einfach aufgebaut. Die Gleichrichtung geschieht über 3A Brücken-(Grätz-)gleichrichter, und die Spannungsstabilisierung über einen positiven Spannungsregler 78Sxx mit dem jeweiligen Wert und ein- und ausgangsseitig einen Elektrolytkondensator. Zum Schutz für die angeschlossenen Geräte ist je eine Glasfeinsicherung kombiniert mit einer LED in verschiedenen Farben, die anzeigen, ob die Spannung anliegt.

Die Schaltung ist, so denke ich, durch die beigefügte Zeichnung und Diagramme mit erläutert.

Das vorhandene Niederspannungskabel hat genug Adern zum Anschluß von MB 02 und Spectrum.

Für die Wärmeableitung habe ich einen absaugenden 12V-Lüfter oben auf das Netzteil gebaut. Ohne Lüfter kann man das Gehäuse gerade noch anfassen, ohne sich die Finger zu verbrennen. Beim Dauertest auf dem Treffen in Urmond (NL) ist mit Lüfter fast keine Erwärmung zu merken.

Bauteil	5V	7,5V	12V
Gleichrichter	alle Brückengleichrichter 40V / 3A		
Elektrolytkondensator je 35V und zweimal	in Mikrofarad 2200	2200	1000
Spannungsregler	78S05	78S75	78S12
LED mit integriertem Widerstand	rot	grün	gelb
Glasfeinsicherung	3.15 A	3,15 A	1,25 A

Ich denke, daß für Bastler ist ein Nachbau leicht zu realisieren. Wer kein +3-Netzteil besitzt, kann es auch mit anderen Trafo und Gehäuse aufbauen, wie ich es mir noch anfertigen will, für den Spectrumbetrieb zu Hause.
Die Kosten für die verwendeten Teile beim Einbau ins +3-Netzteil sind etwa 15,50 Euro bei Conradbestellung, der Lüfter ist extra.
Bei Anfragen ist meine Adresse:

Norbert Opitz , J. F. Böttger Str. 7
06886 Lutherstadt Wittenberg
Tel.: 03491 401573 (werktags nach 16 Uhr)
Mail: NorbertOpitz.Wittenberg@t-online.de

ZiLOGs Rettungsanker: der eZ80

Seit einiger Zeit ist er tatsächlich (zum Preis von 11 US-Dollar ab einer Abnahme von 10.000 Stück) zu kaufen, der eZ80-Prozessor, der die legendäre Z80-Tradition der Firma ZiLOG[1] fortsetzen soll. Vor allem soll der vor zweieinhalb Jahren angekündigte [2] und seitdem von vielen sehnsüchtig erwartete 50 MHz schnelle Embedded-Controller dazu beitragen, die angeschlagene Firma ZiLOG wieder in die Gewinnzone zu bringen. Die Halbleiterfirma hat die drohende Pleite[3] gerade noch einmal abwenden können und jetzt im Rahmen des amerikanischen Konkursrechtes unter Gläubigerschutz nach Chapter 11 eine Umgestaltung vollzogen.
Der eZ80-Webserver (EZ80190A) ist aber weit mehr als eine Wiedergeburt des alten 8-bittigen Z80-Designs. So unterstützt der CPU-Kern neben den Z80- und Z180-kompatiblen Befehlen neue Adressmodi und 24-bittige Register, um 16 MByte linear adressieren zu können. Ihm zur Seite steht im eZ80 jede Menge Peripherie (siehe Steckbrief[4]). Für seine "Webserver"-Fähigkeit liefert ZiLOG einen TCP/IP-Stack, der 15 gängige Protokolle (bis auf IPv6) unterstützt. Derzeit wird der Stack noch als externes Software-Paket dazugeliefert, die nächste eZ80-Generation (F91) soll ihn dann teilweis intern "eingebrannt" bekommen (per integriertem Flash-Speicher von 128 oder 256 KByte). Dieser F91-Chip, der für das Jahresende 2002 vorgesehen ist, soll außerdem eine integrierte Netzwerkschnittstelle bekommen, die beim aktuellen eZ80 noch extern mit Chips von Fremdanbietern zu realisieren ist.
Der eZ80 ist primär als ein Webservertauglicher Embedded Controller gedacht, der auf der riesigen Infrastruktur vorhandener Z80-Software zum Messen, Steuern, Regeln etc. aufbauen kann. Aber es frohlocken auch die alte Z80-Kämpen, etwa die Schneider-CPC-Fangemeinde, die damit einen CPC neuerer Generation (cpcng[5]) als Windows-freien "Familiencomputer" im Rahmen eines offenen Projektes gestalten wollen. Da kann man nur viel Erfolg wünschen.

URL dieses Artikels: ***http://www.heise.de/newsticker/data/as-28.05.02-000/***

Links in diesem Artikel:
[1] *http://www.zilog.com;* [2] *http://www.heise.de/newsticker/data/as-20.09.99-001/;* [3] *http://www.heise.de/newsticker/data/wst-29.11.01-003/;* [4] *http://www.heise.de/newsticker/data/as-28.05.02-000/#eZ80;* [5] *http://www. cpcng.de*; [6] *mailto:as@ct. heise. de*
